

SEG3310 - Object Oriented Analysis, Design and Programming

Topic 11 C# (C-Sharp)

What is C#?

- C# (pronounced "C sharp") is an object-oriented language that is used to build applications for the Microsoft .NET platform
- C# is designed by Microsoft to combine the power of C/C++, Java and the productivity of Visual Basic
- The goal of C# and the .NET platform is to shorten development time
 - by allowing developers to spend their time working on the application logic instead of low level programming details

SEG4210 - Topic 11 - C#

2

Common Language Runtime

- **C# code compiled to an Intermediate Language (IL) which then runs in the Common Language Runtime (CLR)**
 - just like Java is compiled to Java byte code which then runs in the Java Virtual Machine (JVM)
- **The Common Language Runtime (CLR) provides a solid foundation for developers to build various types of applications**
 - such as web services, mobile code application etc
- **The CLR provides the following benefits for application developers:**
 - simplified development
 - integration of code written in various languages
 - assembly-based deployment
 - a better versioning of reusable components
 - code reuse through implementation inheritance
 - etc.

SEG4210 - Topic 11 - C#

3

Compiling C# Source Code

- C# file names have the extension `.cs`
- To create the IL file, we need to compile the `.cs` file using the `csc` (using the command line), followed by the name of the source file
- The result is a file with the same name but the `.exe` extension, which is called an **assembly**
- The assembly file contains all the information that the common runtime needs to know to execute the program
- We can also create multi-file assemblies using an assembly linker, see the following link for more details:

<http://longhorn.msdn.microsoft.com/lh/sdk/ndp/tsk/howto/build/multifile/assembly.aspx>

SEG4210 - Topic 11 - C#

4

Features of C#

- C# syntax is very similar to Java (and thus C++)
- C# features are very similar to Java
 - Object-orientation with single inheritance
 - Support of interfaces
 - No pointers (except for unsafe code)
 - Exceptions
 - Threads
 - Namespaces (like Packages)
 - Strong typing
 - Garbage Collection
 - Reflection
 - Dynamic loading of code

SEG4210 - Topic 11 - C#

5

A First C# Program: 'Hello World'

```
using System;

public class HelloWorld
{
    public static void Main(string[] args)
    {

        // This is a single line comment
        /* This is a
           multiple
           line comment */

        Console.WriteLine("Hello World");
    }
}
```

SEG4210 - Topic 11 - C#

6

About the Hello World program

- C# is case sensitive
- using System consists of importing the System package
- Entry point of C# program must be called Main and not main
 - .NET naming conventions use a capital letter for method names
- C# Main method can have a void parameter as well as an array of strings
- Console.WriteLine is equivalent to System.out.println
- Unlike Java, the file name does not need to be the same as the class name
- The same file can have many classes

Data Types

- **C# supports the following types**
- **Value Types:**
 - primitive types: bool, char, int, short, long, byte, float, double.
 - They can be signed or unsigned (e.g. ulong means unsigned long, etc)
 - Enumeration using the enum keyword
 - Structures using the struct keyword
- **Reference Types: Classes, Interfaces, Arrays and Delegates**
- **Pointers: C# restricts the use of pointers to unsafe code only (see next slide)**

Pointers

- It is possible to have pointer types if the C# code is executing in an unsafe context
- When C# code is executing in an unsafe context, a lot of runtime checking is disabled
 - the program must have full trust on the machine it is running on
- The syntax and semantics for writing pointers is similar to the syntax and semantics for using pointers in C and C++
- To write unsafe code, the unsafe keyword must be used to specify the code block as unsafe
 - and the program must be compiled with the /unsafe compiler switch

General Statements

- **Conditional statements**
 - if – else
 - switch - default
- **Loop statements**
 - while loops
 - do-while loops
 - for loops
 - foreach loops
- **Note that foreach is very useful for traversing collections.**
Example:

```
string[] alphabet = {"a", "b", "c", "d", "e"...};
foreach(string str in alphabet)
    Console.WriteLine(str + " is a letter of the alphabet");
```

The Class Object

- C# has a single rooted class hierarchy where all classes in C# are subclasses of System.Object
 - the same way all Java classes are subclasses of java.lang.Object
- Java and C# Object share some common methods such as the toString() (ToString() in C#) method
- C# uses aliases to refer to some class names. For example the class System.Object can be written object (with small 'o')
 - string is an alias for System.String

Boxing and Unboxing

- Value types (struct, enum, int..) can be transformed into reference types using Boxing and Unboxing
- **Boxing:** the following assignment wraps up the value 3 into an object

```
object obj = 3;
```
- **Unboxing:** this assignment unwraps the value

```
int x = (int) obj;
```
- This is similar in principle to Java wrapping classes

Rectangular and Jagged Arrays

- C# allows both jagged and rectangular arrays to be created
- Jagged arrays are the same as Java arrays. They are arrays of arrays. Example:

```
int [][] array = new int [3][4]; // creates 1+3 = 4 arrays
```
- Rectangular arrays are more efficient but less flexible. The arrays have to be of the same dimension:

```
int [,] array = new int [3, 4]; // creates only 1 array
```

The System.String Class

- This is the same as the Java String class
- Strings can be concatenated with +
- They can be indexed: `s[i]`
- String length: `s.Length`
- Strings are reference types => reference semantics in assignments
- Their values can be compared with `==` and `!=`
- Class String defines many useful operations: `CompareTo`, `IndexOf`, `StartsWith`, `Substring`, etc

Classes

- C# classes are similar to Java classes
- C# allows the following class members (most of them are explained in the coming slides)

Constructors
Destructors
Fields
Methods
Properties
Indexers
Delegates
Events
Nested Classes

Example of a class in C#

```
// The class Example
using System;
class Example
{
    private string myString;

    // Constructor
    public Example(string myString)
    {
        this.myString = myString;
    }

    // Instance Method
    public void printString()
    {
        Console.WriteLine(myString);
    }
}

// Creating objects of the class Example
// Program start class
class ExampleClass
{
    // Main begins program execution
    public static void Main()
    {
        // Instance of Example
        Example obj = new Example("Message");

        // Call Output class' method
        obj.printString();
    }
}
```

The is operator

- The C# `is` operator is completely analogous to Java's `instanceof` operator.
- The two following code fragments are equivalent.

C# Code

```
if(x is MyClass) MyClass mc = (MyClass) x;
```

Java Code

```
if(x instanceof MyClass) MyClass mc = (MyClass) x;
```

Access Modifiers

- C# access modifiers
 - **public**: accessible by any other class
 - **private**: accessible only by the class in which it is declared
 - **protected**: accessible only by the class in which it is declared, as well as any derived classes
 - **internal**: accessible only from within the same assembly
 - **protected internal**: accessible only from within the same assembly and the derived classes of the containing class
- If the modifier is not specified then it is considered **private** by default (similar to C++)

Constructors

- Constructors in C# works similarly to those in C++ and Java.
 - Multiple constructors can be defined
- C# also provides a mechanism to initialize static members of a class.
 - This is done using a static constructor:

```
static MyClass() {  
    ...  
}
```
- Static constructors are implicitly public and necessarily take no input parameters

Destructors

- They are similar to the concepts of finalizers in Java.
- They are called before the object is removed by the garbage collector
- Destructors (as well as Java finalizers) should be avoided
 - because there is no way to control the order of finalization of inter-related objects

- Syntax:

```
class Test {  
    ~Test() {  
        ... finalization work ...  
        // automatically calls the destructor of the superclass  
    }  
}
```

Fields Modifiers

- Fields modifiers include all access modifiers seen earlier
- To declare constants in C#
 - the const keyword is used for compile time constants
 - while the readonly keyword is used for runtime constants
- Example of a run time constant

```
static readonly uint var = (uint) DateTime.Now.Ticks;
```
- Recall that to declare constants in Java the final keyword is used in both cases

Methods Modifiers

- There are ten modifiers that can be used with methods
- Four of these are the access modifiers seen before
- The rest are the following:
 - abstract**: determines abstract methods (similar to Java)
 - static**: C# supports static fields and methods just like Java
 - new**, **virtual**, **override**: used for method overriding, we will see these when we cover inheritance
 - extern**: used to call methods that are written in other languages, this is similar to Java native methods
- C# supports method overloading just like Java
- Unlike Java, C# supports also operator overloading (similar to C++)

Parameter Passing

- The parameter modifiers ref specifies that a parameter is passed by *reference* (similar to VB)
- If no modifier is specified then the parameter is passed by *value*
- C# defines output parameters to specify parameters that return values. This is done using the out modifier
 - this is useful if multiple values are returned from a method

Variable Number of Parameters

- C# uses the params modifier to pass an arbitrary number of arguments to a method
- It is **not** necessary to place these arguments into an array before calling the method:

```
using System;  
class ParamsTest {  
    public static void Print(string title, params int[] args) {  
        Console.WriteLine(title + " :");  
        foreach(int num in args)  
            Console.WriteLine(num);  
    }  
    public static void Main(string[] args) {  
        Print("First 4 positive numbers", 0, 1, 2, 3);  
    }  
}
```


Properties

- Properties are used to represent getter/setter methods in a more flexible way
 - this concept exists in Delphi and Visual Basic
- It is possible to create read-only, write-only or read and write properties
- Java does not have a mechanism for implicitly defining getter and setter methods
 - it is up to the programmer to define these methods

Example of C# properties

```
using System;

public class Person {
    private string name;
    private int age;

    public Person(string name){
        this.name = name;
    }

    //read-only property for name
    public string Name{
        get{
            return name;
        }
    }

    //read-write property for age
    public int Age{
        get{
            return age;
        }
        set{
            age = value;
        }
    }

    public static void Main(string[] args){
        Person pers = new Person("Bob");
        pers.Age = 60;
        Console.WriteLine("Name: " +
            pers.Name + " " + pers.Age);
    }
}
```

Indexers

- An indexer is a special syntax for overloading the [] operator for a class
 - Java does not support this feature. C++ does by overloading the [] operator
- An indexer is useful when a class is a container for another kind of object
- It is also possible to create indexers that allow multidimensional array syntax
- Indexers can also be overloaded

Example of using indexers

```
public class Building
{
    Floor[] allFloors;
    // the name of an indexer is always this
    public Floor this [int index] {
        get {
            return allFloors[index];
        }
        set {
            if (value != null) {
                allFloors[index] = value;
            }
        }
    }
    ...
}
```

How to use it in a program:

```
Building aBuilding = new Building(parameters);
aBuilding[10] = new Floor(parameters);
```

Inheritance

- C# does not support multiple inheritance just like Java
- C# does not support C++ different kinds of inheritance
 - private and protected
- The following example shows a class B
 - that inherits from A and
 - implements the interface IComparable

```
using System;

class B : A, IComparable {
    int CompareTo() {}

    public static void Main(String[] args) {
        Console.WriteLine("Hello World");
    }
}
```

Calling the Superclass Constructor

- The operator base is used to call a constructor of the superclass as shown in the following example:
- ```
public MySubClass(string s) : base(s) {

}
```
- C# explicitly enforces the call to the superclass constructor before the class constructor (similar to C++)
  - base is similar to super in Java

## Overriding Methods and Polymorphism

- In order to be overridden by a derived class, a method must be declared either abstract or virtual
- In addition, developers of a subclass
  - can decide to either explicitly override the virtual method by using the override keyword
  - or explicitly choose not to by using the new keyword instead
    - if you choose not to override a virtual method is like you create a new method that is independent from the superclass method

## Interfaces

- Interfaces in C# are similar to Java interfaces
  - they are denoted using the keyword interface
- C# interfaces contain only abstract methods
  - In Java, interfaces may contain variables
- In .NET naming conventions, interface names start with an upper-case "I" (as in ICloneable)
- As shown previously, the class that implements an interface does not use the Java keyword implements
  - It is used as a normal superclass

## Unextendable Classes

- Both Java and C# provide mechanisms to specify that a class cannot be used as a superclass
- In Java this is done by preceding the class declaration with the final keyword while
- In C# this is done by preceding the class declaration with the sealed keyword

## Namespaces

- A C# namespace is a way to group classes and is used in a manner similar to Java's package construct
- C# namespaces are similar to C++ namespaces and syntax
- Unlike Java, C# namespaces do not dictate the directory structure of source files in an application
- Namespaces can be nested similar to Java packages

## Namespaces (cont.)

| C# Code                                                                                                                       | Equivalent Java Code                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <pre>namespace GUI {<br/>    public class MyClass {<br/>        int x;<br/>        public void m(){...}<br/>    }<br/>}</pre> | <pre>package GUI;<br/>public class MyClass {<br/>    int x;<br/>    public void m(){...}<br/>}</pre> |

## Exceptions

- Exceptions in C# and Java share a lot of similarities.
- Both languages support the use of the try and catch blocks for handling thrown exceptions
  - and the finally block for releasing resources before leaving the method
- Both languages have an inheritance hierarchy where all exceptions are derived from a single Exception class
- Exceptions can be caught and rethrown

## Exceptions (cont)

- Checked exceptions are typically used to indicate to a calling method that the callee generated an exception
- Java supports checked exceptions
- In C#, all exceptions are unchecked and there is no support to the throws clause
  - a major drawback is that it is not possible for a programmer to know what exceptions to catch in their code

## Example of a user defined exception

```
using System;

class MyException: Exception {

 public MyException(string message): base(message){ }

 public MyException(string message, Exception
 innerException): base(message, innerException){ }
}
```

## Synchronization in C#

- C# provides the lock statement which is semantically identical to the synchronized statement in Java
  - to ensure that only one thread can access a particular object at a time
- Example

```
public void Withdraw(int num){
 lock(this){
 if(num < this.amount)
 this.amount -= num;
 }
}
```

## Collections

- The C# collections framework consists of the classes in the **System.Collections** namespace
- Java collections framework is more sophisticated than that available in .NET via C#
- **System.Collections** contain several interfaces and abstract classes
  - such as IList, IEnumerable, IDictionary, ICollection, and CollectionBase
  - which enable developers to manipulate data structures independently of their implementation
- The **System.Collections** namespace also contains some concrete implementations such as ArrayList, Queue, SortedList and Stack

## Reflection

- Reflection in C# is done at the assembly level while in Java it is done at the class level
- Since classes are stored in assemblies then we need to load the assemblies
  - while in Java one needs to be able to load the class file for the targeted class
- C# Reflection library is found in **System.Reflection**;

## File I/O

- C# and Java support performing I/O via Stream classes
  - C# IO classes are found in **System.IO**
  - The following example displays the content of the file "input.txt"
- ```
using System;
using System.IO;
public class FileIOExample {
    public static void Main(string[] args){
        FileStream inputFile = new FileStream("input.txt", FileMode.Open);
        StreamReader sr = new StreamReader(inputFile);
        String str;

        while((str = sr.ReadLine())!= null)
            Console.WriteLine(str);

        sr.Close();
    }
}
```

References

- **Microsoft MSDN**
<http://msdn.microsoft.com/>
- Dare Obasanjo, "A Comparison Of Microsoft's C# Programming Language To Sun Microsystems' Java Programming Language"
<http://www.25hoursaday.com/CsharpVsJava.html#checked>

SEG4210 - Advanced Software Design and Reengineering

TOPIC 19 Garbage Collection Algorithms

What is Garbage Collection?

- The automatic management of dynamically allocated storage
- Automatically freeing objects that are no longer used by the program
- Refer to Richard Jones garbage collection web site for a rich set of online resources:
<http://www.cs.kent.ac.uk/people/staff/rej/gc.html>

Why Automatic Garbage Collection?

- It is not generally possible to immediately determine when a shared object becomes no longer 'in use'
 - So *explicit* memory management is often difficult to perform
- Software engineering perspective:
 - Garbage collection increases the abstraction level of software development
 - Decreases coupling among the system modules
 - Frees software developers from spending a lot of time managing memory
 - Eliminates many memory-management bugs

Terminology

- **Stack:**
 - A memory area where data is
 - pushed when a procedure is called
 - and popped when it returns
 - Contains *local* variables (which must be cleaned up)
- **Heap:**
 - A memory area where data can be allocated and deallocated in *any order*
 - Functions like 'malloc' and 'free', or the 'new' and 'delete' operators allocate and free data in the heap
 - In most good OO program all objects are allocated on the heap

Terminology (cont.)

- **Root Set:**
 - A set of objects that a program always has direct access to
 - E.g. global variables, or variables in the main program (stored on the program stack)
- **A Heap Object (also called a cell or simply object):**
 - An individually allocated piece of data in the heap
- **Reachable Objects:**
 - Objects that can be *reached transitively* from the root set objects

Terminology (cont.)

- Garbage:
 - Objects that *are unreachable* from root set objects but *are not free* either
- Dangling references:
 - A reference to an object that was deleted
 - May cause the system to crash (if we are lucky!)
 - May cause more subtle bugs
- Mutator:
 - The user's program
 - (often contrasted with the 'collector')

Two Main Types of Algorithms

- Reference counting
 - Each object has an additional field recording the number of objects that point to it
 - An object is considered garbage when zero objects point to it
- Tracing
 - Walk through the set of objects looking for garbage

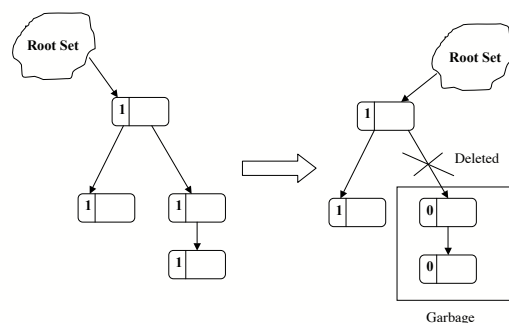
Reference Counting

- Each object has an additional field called the *reference count*
- When an object is first created, its reference count is set to one
- When any other object (or root object) is assigned a reference to that object
 - **then its reference count is incremented**
- When a reference to an object is deleted or is assigned a new value
 - **the object's reference count is decremented**

Reference Counting

- Any object with a reference count equal to zero can be garbage collected
- When an object is garbage collected,
 - ***Any object it refers to has its reference count decremented***
 - The garbage collection of one object may therefore lead to the immediate garbage collection of other objects

Reference Counting Example



Pros and Cons of Reference Counting

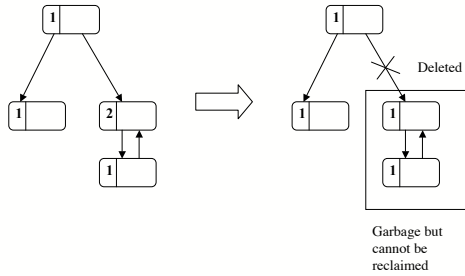
Pros:

- The garbage collector is executed along with the mutator
- Free memory is returned to free list quickly

Cons:

- The reference counts must be updated every time a pointer is changed
- We need to save an additional field for each object
- Unable to deal with cyclic data structures (see next slide)

Cyclic Data Structure



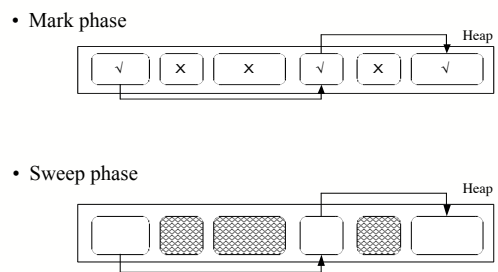
Tracing Algorithms

- Used more widely than reference counting
- Visit the heap objects and determine which ones are not longer used
- Tracing algorithms differ according to:
 - Whether all objects are visited or not
 - Whether they use the heap in an optimal way or not
 - Whether the collector is executed in parallel with the mutator or not
 - The duration of the pauses that the mutator undergoes when the algorithm is executed

Mark-Sweep Algorithm

- The first tracing algorithm
- Invoked when the mutator requests memory but there is insufficient free space
- The mutator **is stopped** while the mark-sweep is executed
 - This is impractical for real-time systems
- Performed in two phases:
 - **Mark phase:** identifies all reachable objects by setting a mark
 - **Sweep phase:** reclaims garbage objects

Mark-Sweep Example



Pros and Cons of Mark-Sweep

Pros:

- Cyclic data structures can be recovered
- Tends to be faster than reference counting

Cons:

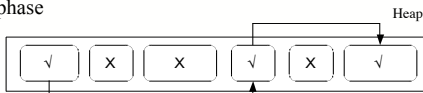
- Mutator must stop while the algorithm is being performed
- Every reachable object must be visited in the mark phase and every object in the heap must be visited in the sweep phase
- Causes memory fragmentation

Mark-Compact algorithm

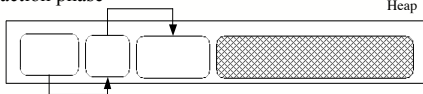
- Similar to the mark-sweep algorithm except that it does not cause memory fragmentation
- Two phases:
 - **Mark phase:** identical to mark-sweep
 - **Compaction phase:**
 - marked objects are compacted
 - Reachable objects are moved forward until they are contiguous.

Example

- Mark phase



- Compaction phase



Pros and Cons of Mark-Compact

Pros:

- Eliminates fragmentation problem

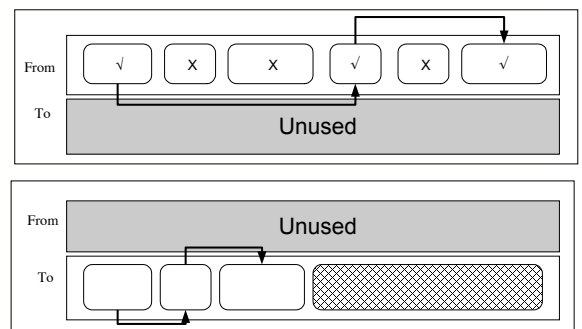
Cons:

- Mutator must stop while the algorithm is being performed
- Several passes over the objects are required to implement the compaction

The Copying Algorithm

- The Copying algorithm splits the heap into two equal areas called from-space and to-space
- The mutator works in the from-space area
- The algorithm visits the reachable objects and copies them contiguously to the to-space area
 - Objects need to be traversed only once
- Once the copying is completed, the to-space and from-space switch roles

Example



Pros and Cons of The Copying Algorithm

Pros:

- Eliminates Fragmentation
- Copying is very fast
 - provided the percentage of reachable objects is low
 - It only visits reachable objects

Cons:

- Mutator must stop while the algorithm is being performed
- The use of two areas doubles memory space
- Impractical for very large heaps

Incremental Tracing Algorithms

- The previous tracing algorithms are also known as **Stop-The-World algorithms**
 - They stop the execution of the mutator in order to start performing
- Incremental algorithms (also called parallel algorithms) run concurrently with the mutator
 - Can be used in systems with real-time requirements

Incremental Tracing Algorithms (cont.)

- Garbage collection can be executed as a thread
- Reference counting can be considered as an incremental algorithm
 - However, most languages do not use it due to its numerous disadvantages
- There exists an incremental version for some of the non-incremental algorithms seen before
 - E.g. Baker's copying incremental algorithm

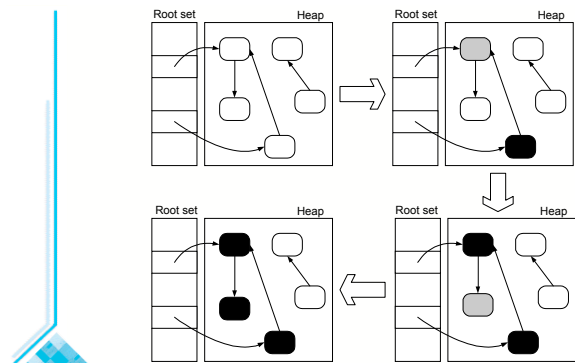
Tricoloring Algorithm

- It is an incremental algorithm based on coloring objects
- An object can have one of three colors
 - White:
 - Initial state of all objects
 - Not visited
 - If it remains white at the end, then it can be collected
 - Black:
 - Visited by the collector, so confirmed reachable
 - And has no direct references to White objects
 - Grey:
 - Visited but *not all the objects it refers to have been visited*
 - *When this set becomes empty, all remaining white objects can be destroyed*

Tricoloring Algorithms (cont.)

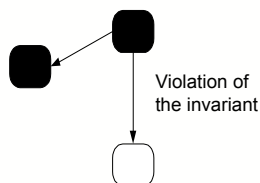
- The steps of the algorithm are:
 - Start with all objects white (not visited)
 - Mark root objects grey
 - While there are grey objects
 - take a grey object
 - mark its children grey, then mark it black
 - At the end, all white objects are garbage

Tricoloring Marking Example



Tricoloring Invariant

There must not be a pointer from a black object to a white object



The GC algorithm on its own can guarantee this

- But the mutator may violate it unless we are careful

Preventing Tricolor Variant Violation

- Two ways to prevent violation of the tricoloring invariant
 - Both slow down the mutator slightly
- Using a **write barrier**:
 - Prevent the mutator from making a pointer in a black object to a white object.
 - If this is attempted, mark the black object grey
- Using a **read barrier**:
 - Any attempt to access a white object proves it is reachable, so mark it grey

Pros and Cons of Incremental Algorithms

Pros:

- The mutator is not stopped
—just paused for short periods)

Cons:

- Hard to synchronize the mutator with the garbage collector
- Take more execution time because of the barriers unless specialized hardware is used
- Hard to debug

Generational Garbage Collection

- The previous tracing algorithms execute on all objects
- Generational algorithms improve this by dividing objects into generations
- Based on the empirical observation that: **Most objects die young**
- Garbage collection is then executed more frequently on objects that are likely to be garbage: new objects

Generational Garbage Collection (cont.)

- Object lifetime is measured based on the amount of heap allocation that occurs between the object's creation and deletion
- The heap is divided into several areas, called **generations**, that hold objects according to their age
- Areas containing newer objects are garbage collected more frequently
 - Resulting in less pause times for the mutator
 - This process is called *minor collection*

Generational Garbage Collection (cont.)

- Areas containing older objects are garbage collected less frequently
 - This process is called *major collection*
- After an object has survived a given number of collections
 - It is **promoted** to a less frequently collected area
- Choosing the right number of generations and the promotion policies can be a problem
 - Some objects can be 'tenured', meaning the garbage collector never looks at them

Intergenerational Pointers

- An intergenerational pointer is a pointer from an old generation object to a newer generation object
- Intergenerational pointers need to be tracked
- If these pointers are not tracked then
 - a young object may be garbage collected if it is only referenced by older objects

Garbage Collection and Java

- The 1.0 and 1.1 JDKs used a mark-sweep collector,
 - which causes memory fragmentation
- Allocation and deallocation costs were high
 - since the mark-sweep collector had to sweep the entire heap at every collection

Garbage Collection and Java (cont.)

- In HotSpot JVMs (Sun JDK 1.2 and later), the Sun JDKs used a generational collector
- The copying algorithm is used for the young generation
 - The free space in the heap is always contiguous
 - Both allocation and deallocation costs have improved compared to previous versions
- Mark-Compact is used for old objects

References

- “Garbage Collection : Algorithms for Automatic Dynamic Memory Management”
by Authors: Richard Jones , Rafael D Lins

SEG4210 - Advanced Software Design and Reengineering

TOPIC 20 Secure Software Development

Security: A combination of factors

Dependability

- The software runs as intended under all circumstances, even when under attack

Trustworthiness

- The software contains no vulnerabilities that can be exploited by an attacker

Survivability

- Resists attacks (protects itself from them actively)
- Tolerates attacks (continues to provide service while being attacked)
- Recovers from attacks, that it wasn't able to resist or tolerate, as quickly as possible and with as little damage as possible

General types of attacks

Unauthorized access or interception

- E.g. to steal data, identity or money, modify data, etc.

Overstepping authority (accidental or on purpose)

- A legitimate user does things they shouldn't

Adding a payload

- Inserting viruses, spyware, bots, etc.

Vandalism and corruption

- Making a system not appear or behave as it should

Spoofing

- Redirecting legitimate users to an illegitimate place

Denial of service

- Overloading network or computational resources so legitimate users can't use the system

Motivations of attackers

Financial gain

- E.g. cracking into bank accounts
- E.g. theft of identities that can be sold

Achieving personal objectives

- E.g. granting oneself a pilot's license
- E.g. Building a collection of pirated movies

Fun, entertainment, challenge or bragging rights

Revenge / anger / hatred

Political / military

- private, radical group or state sponsored

Some thoughts on attack frequency

A significant proportion of successful attacks are by 'insiders'

- E.g. employees committing fraud
- Physical security can be breached
 - Watching password entry over-the-shoulder, reading written passwords, accessing the physical disk or RAM, bypassing the network

Much attacking today is automated: Botnets

Attackers may try millions of random attacks until they find a 'weak link'

- They will only keep attacking one target if it is extremely valuable

Systems thinking

A system is only as secure as its weakest link

- Often this is the
 - Operating system
 - Reused components
 - Network
 - Human
 - Paper records
 - Hardware

So analyse every possible aspect of the system for its impact on security

Cryptography as one key to security

Beware: cryptography is only one tool in security

- Some people assume it is the only or main tool

Private key cryptography

- Sender and recipient know the secret key and algorithm

Public key cryptography

- You encrypt using the public key published by the recipient
- The result can only be decrypted using a mathematically related private key
- Cracking relies on factoring extraordinarily large numbers
 - Infeasible to do this quickly, although often can be done
 - The more 'bits' in the key, the more computer power needed

Attacks on cryptographically protected systems - 1

On-line

- If the key is related to a human-created non-random password, then try common password choices
 - Dictionary words ("dictionary attacks")
 - Passwords the user has used on other systems

Off-line

- Getting a sample of the data and using a dedicated computer to algorithmically try combinations
- For a random password and good algorithms, an attack has to be exhaustive, making it very hard

Attacks on cryptographically protected systems - 2

Social engineering

- Tricking someone to reveal a key (e.g. phishing)

Weak password-resetting protocols

Man-in-the-middle

- Inserting software that will relay cryptographic keys before they are used

Keystroke logging

Attacks on cryptographically protected systems - 3

There are many 'hackers' tools available on the Internet

- E.g. for doing dictionary attacks
- Try these against your own system to see how secure it will be

Secure passwords - 1

Note that a password is rarely as secure as the number of bits in a cryptographic key

- Not as long
- Not as random

Nevertheless encourage / require users to use

- Longer passwords (8+ characters)
- Combination of character types
 - Lower/upper case, numbers, special characters
- Minimal duplicate characters
- No numbers at the end
- No password similar to a recently used password
- Not containing dictionary words

Secure passwords - 2

Back up password protection with other schemes

- Slow then block access after multiple failed attempts
- Detect and prevent automated entry
 - e.g. very quick response to prompt
- Add extra checks when access comes from unexpected place or time
 - Non-normal IP address range
 - Late at night
- Check ability to answer pre-saved questions
 - But beware of those that reveal personal information
- CAPTCHAS

Biometrics

Various types based on recognition of

- Fingerprints
- Irises
- Palm pattern
- Face
- Voice
- Signature

All have some risk of false positive and false negative

- Should be backed up by other schemes for critical applications

Hardware devices: The good and the bad

Devices to increase security

- Devices with smart chips such as smart cards or USB dongles
 - Physical presence of device lends credence to authenticity
 - But they can be stolen or forged, so they should not be fully relied on

Risks from devices

- E.g. USB keys or disks that harbor viruses

Principles to increase security - 1

Understand the motivations and methods of attackers

Avoid the most common design and coding mistakes

- Discussed later

The more 'benefit' for the attacker, the more capable an attacker to expect

- So invest more in security when stakes are higher

Increase the expense of attacking

- E.g. ensure it take more time by using more bits in cryptographic keys

Principles to increase security - 2

Increase attacker uncertainty

- Hide and randomize names and locations of resources
 - Obfuscation
- Avoid clear feedback that could give clues to an attacker about whether they are succeeding or not
- Use honeypots
 - Targets that take work to attack, look as though they have valuables, but are fake

Isolate from network if possible, or make invisible on network

Principles to increase security - 3

Incorporate adequate monitoring and logging so attacks can be detected, tracked and forensically analysed

Limit and control the number of legitimate users

Grant only needed privileges to users

- Principle of least privilege
- Information access on 'need to know' basis
- Have unused privileges expire

Ensure users know acceptable and unacceptable practice

Principles to increase security - 4

Make secure practices usable

- Balance requirements and risks
 - Requirement to use 'strong' passwords
 - Requirement to change passwords
 - Requirement to use different passwords on each system
- vs
- Risk that people will write down passwords

Automatically dispose of data that is no longer needed

- The more retained data, the more loss in case of a breach and the more attractive to attackers

Principles to increase security - 5

Secure both software and IT infrastructure

- Examples of securing IT infrastructure
 - Require laptops (or all computers) to have data on board encrypted at all times
 - Use 'call home' tools to track stolen computers
 - Force maximum use of anti-virus software and firewalls
 - For guest use of wireless network, have time-limited individual accounts on a separate subnet
 - Disallow arbitrary software installation
 - Disallow attachment of removable media
 - Automatically patch all machines

Principles to increase security - 6

- Close unneeded TCP ports
- Deploy a VPN for access to network
- Back up vigorously, but secure the backups
- Update cryptographic and other techniques as vulnerabilities are revealed
 - E.g. avoid WEP on a wireless network
- Force new systems to have the securest settings enabled
- Use sandboxes and virtualization to 'contain' security breaches
- Securely erase / destroy old systems
- Constantly monitor for intrusion
- Employ an IT security officer

The CWE/SANS Most Dangerous Programming Errors

Reference: <http://www.sans.org/top25errors/>

CATEGORY: Insecure Interaction Between Components

- Improper Input Validation
 - E.g. allowing arbitrary html to be entered
 - E.g. allowing violation of input constraints
- Improper Encoding or Escaping of Output
 - E.g. hackers may be able to get one system to output a command that will be executed by another
- Failure to Preserve SQL Query Structure (aka 'SQL Injection')
 - E.g. a data string that ends an insert, followed by 'Delete table'
- Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
 - E.g. Allowing a script from an arbitrary linked site to change contents from your site

The Most Dangerous Programming Errors 2

- Failure to Preserve OS Command Structure
 - 'OS Command Injection
- Cleartext Transmission of Sensitive Information
- Cross-Site Request Forgery (CSRF)
 - It looks to a server that the request is coming from a page it served
- Race Condition
 - Applications behave unpredictably, giving hackers information
- Error Message Information Leak

The Most Dangerous Programming Errors 3

CATEGORY: Risky Resource Management

- Failure to Constrain Operations within the Bounds of a Memory Buffer
 - AKA “Buffer Overflow Errors”
- External Control of Critical State Data
 - E.g. cookies, files, etc. that can be manipulated by a hacker
- External Control of File Name or Path
 - E.g. If the hacker gets to choose a file name he can type “../” to walk up the directory hierarchy
- Untrusted Search Path
 - The application goes to a location of the hacker’s choosing instead of where intended

The Most Dangerous Programming Errors 4

- Failure to Control Generation of Code
 - ‘Code Injection’
 - Many apps generate & execute their own code
- Download of Code Without Integrity Check
 - The hacker’s code gets downloaded instead
- Improper Resource Shutdown or Release
 - E.g. a file is left open, then accessed by a hacker
- Improper Initialization
 - A hacker may be able to initialize for you, or see data from a previous use
- Incorrect Calculation
 - Hackers take control of inputs used in numeric calculation

The Most Dangerous Programming Errors 5

CATEGORY: Porous Defenses

- Improper Access Control (Authorization)
- Use of a Broken or Risky Cryptographic Algorithm
 - E.g. WEP
- Hard-Coded Password
- Insecure Permission Assignment for Critical Resource
- Use of Insufficiently Random Values
- Execution with Unnecessary Privileges
- Client-Side Enforcement of Server-Side Security

Security in the software lifecycle

Requirements

- Ensure security needs are identified and quantified
- Threat and risk analysis

Formal specification of security properties

Design

- Follow proper design practices

Testing and quality assurance

- Rigorously inspect and test all security mechanisms
- Employ people to act as hackers to try to break system

Deployment

- Ensure safeguards are properly installed and put into use

Evolution

- Adapt as new threats become known

Privacy: A related but distinct issue

To protect privacy

- Secure personal information about customers, users, employees, etc.
 - Identification data, phone numbers, addresses, account numbers, data of birth, etc.
- Only gather the minimal information needed
 - Also important for security
- Delete the information when not needed
 - Also important for security
- Ensure the user has a right to
 - Know what information is gathered
 - Know what use it is to be put to
 - Review it, change it and delete it as needed
- Only use the information for the stated purpose
- Follow all privacy laws, e.g. PIPEDA

A useful web site on security

From the US government:

- Build security in
 - <https://buildsecurityin.us-cert.gov/daisy/bsi/547-BSI.html>

TOPIC 14

Ruby on Rails

What is Ruby on Rails?

- *Ruby* is an object-oriented programming language
 - Incorporates ideas from Smalltalk, Java, etc.
 - Very pure
- *Rails* is a Ruby framework for creating web applications quickly and easily

The Ruby language - basics

An open-source, dynamic language

- Created by Yukihiro “matz” Matsumoto
 - Originated in 2001, Public release in 2005
- Supported by a large community



Main website:

- www.ruby-lang.org

Rapidly growing library and online resources

- http://rubyforge.org/softwaremap/trove_list.php
- <http://www.saphirsteel.com/The-Little-Book-Of-Ruby>

Store programs in files with extension .rb

- Or run the *irb* interpreter

Until last year, the fastest growing language <http://www.tiobe.com/tpci.htm>

Position Dec 2008	Position Dec 2007	Delta in Position	Programming Language	Ratings Dec 2008	Delta Dec 2007	Status
1	1	=	Java	19.367%	-0.68%	A
2	2	=	C	16.163%	+2.99%	A
3	5	↑↑	C++	10.893%	+3.02%	A
4	4	=	PHP	9.479%	+1.09%	A
5	3	↓↓	(Visual) Basic	9.478%	-0.74%	A
6	8	↑↑	C#	4.643%	+0.65%	A
7	6	↓	Python	4.567%	-0.13%	A
8	7	↓	Perl	3.603%	-0.78%	A
9	10	↑	JavaScript	3.062%	+0.33%	A
10	11	↑	Delphi	3.055%	+0.38%	A
11	9	↓↓	Ruby	2.308%	-0.78%	A

Ruby: Flexibility, simplicity and power

- **Everything is an object**
 - Inherits from the Smalltalk legacy
- **No variable declarations needed**
- **Dynamic typing**
- **'Blocks' allow you to define new control structures**
- **'Mixins' (modules) give power of multiple inheritance without the drawbacks.**
- **Code can be added to a class dynamically**
- **Easy to**
 - Add C code (or code in another language)
 - Extend the language

Defining functions

```
def helloworld
  puts "Hello World"
end
```

```
def hello(arg)
  puts "hello "+arg
end
```

Example runs

```
hello "Tim"
hello ("Tim") # preferred - usually helps prevent bugs
```

A few more details 1

Placeholder substitution – works with `"`, but not with `'`
`puts "hello #{arg} Your argument is #{arg.length} long!"`

Default argument to a method – and concatenation
`def hello(arg="World")
 puts "hello "+arg
end`

Defining an array
`x=[1, 2 ,3]
y= %w(an array of strings with the words of this phrase)`

Sorting - the `!` Means alter (side-effect) the object
`x.sort!`

A few more details 2

To access array elements
`x [1..3] #elements 0 to 3
x [1,3] # elements 0 for 3
x [-3,1] # from 3rd-from-the-end for 1`

Non numeric collections are called hashes
`z = Hash.new
z ['SEG4210'] = 'Advanced Software Engineering'
anotherhash = {'test' => 12334, 'test2' => 98776}`

A Ruby method will always return the value of the last expression evaluated

- Although using `'return'` is good practice for clarity

Use ``` to execute a program in the operating system
``ls``

A few more details 3

Symbols

- E.g. `:all`
- Represent specific language keywords used in various contexts

Iteration

```
@names.each do |name|  
  print name  
end
```

```
For number in [1,2,3] do  
  print number  
end
```

```
while tired  
  sleep  
end
```

Ruby has various other allowed syntaxes, including allowing `{ }`

“Duck typing”

In Ruby you don't declare the type of a variable

- You can put any object in a variable.
- You can determine the type in order to make a decision

```
if @names.respond_to?("join")
```

“if it talks like a duck, it is a duck”

Defining a class

The `initialize` method is used to construct the object

- `@name` is an instance variable
- all instance variables are private by default

```
class Greeter  
  def initialize(name = "World")  
    @name = name  
  end  
  
  def say_hi  
    puts "Hi #{@name}!"  
  end  
end
```

To create an object
`g = Greeter.new("Pat")`

Attributes: To make instance variables public

```
attr_accessor :name, :studentNumber
```

Now I can say in another class

```
g.name="Tim"
```

and

```
g.name
```

I can also separately say

```
attr_reader :studentNumber
```

```
attr_writer :age
```

Creating a subclass

```
def specialGreeter < greeter
```

To call the same method with same arguments in a superclass

```
super
```

Class variables (static variables)

```
@@defaultSize = 5
```

Introspection

```
Greeter.instance_methods
```

```
["method", "send", "object_id", "singleton_methods", "say_hi",  
  "__send__", "equal?", "taint", "frozen?", "instance_variable_get",  
  "kind_of?", "to_a", "hello", "instance_eval", "type",  
  "protected_methods", "extend", "eql?", "display",  
  "instance_variable_set", "hash", "is_a?", "to_s", "class", "tainted?",  
  "private_methods", "untaint", "id", "inspect", "dello", "==" , "===" ,  
  "clone", "public_methods", "respond_to?", "freeze", "___id__", "=~",  
  "methods", "nil?", "dup", "instance_variables", "instance_of?"]
```

Note that the "hello" method we defined earlier is here

- It is inherited from class Object

Inspecting an object

```
g.inspect or p(g)
```

Dynamic classes

Ruby is a very dynamic language

- You can modify a class at any time!
- Add new methods, variables, etc.

Modules - have some similarities to interfaces

You can achieve multiple inheritance by simply adding "mixin" modules to several classes

```
module MyModule  
  def func(arg)  
    doSomethingWith arg  
  end  
end  
  
def myclass  
  include MyModule  
  ...  
end
```

Modules also provide **namespaces**

Loading modules from files

```
require (MyModule.rb)
```

Standard modules already loaded in every Ruby class

Comparable, Enumerable
FileTest
GC, Kernel
Math
ObjectSpace, Precision, Process, Signal

Use notation: `X::Y` to refer to class `X` in module `Y`

The Ruby on Rails framework

Makes it easy to develop web applications

- Most manipulate data in a database such as MySQL or Postgres
- Can also manipulate flat files

Integrates with AJAX

Suggested book:

Agile Web Development with Rails

- By
 - Dave Thomas
 - and David Heinemeier Hansson (creator of Rails)
- Second edition Jan 2007, www.pragprog.com

Rails philosophies

Model-View-Controller (MVC)

- Separation of the data from the presentation from the control logic

Don't Repeat Yourself

- E.g. define your database structure in just one place

Convention over Configuration

- Sensible defaults

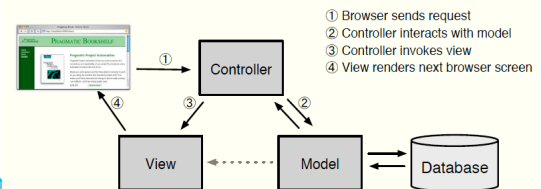
Agility

- The application can be quickly changed

Models MVC

Models

- Whatever is shown in the class diagram plus business rules
- The Ruby model and the Database are separated using ORM (Object Relational Mapping)
- Avoids any need to embed SQL code in an application



Active Record as the basis for the Model

Database tables map to Rails classes

- Rows map to objects

```
require 'active_record'
```

```
# The following is generated from the database and attributes are filled  
# in from database columns
```

```
class Order < ActiveRecord::Base  
end
```

```
order = Order.find(1)  
order.discount = 0.5  
order.save
```

Querying the model in Ruby code

```
Order.find(:all, :conditions => "name='dave'" ).each do  
  order!  
  puts order.amount  
end
```

The View - Three mechanisms

rhtml

- Embedded Ruby code in html, using Erb (embedded Ruby)

rxml

- Construct XML using Ruby code

rjs

- Create javascript in Ruby code to create AJAX applications
- The JavaScript runs in the client, of course

The Controller

Routes http requests to Ruby methods

- Allows you to create people-friendly URLs

Manages Caching

- Database data stored in memory for a speed boost

Manages Sessions

- E.g. a shopping session

Getting started

You need to install

- Ruby
- Rails
- A database
- I leave it to the TA and/or reference manuals to do that

The hard part of application development is now done!!

Really getting started

Create a new empty default application with the command

`rails myapplication`

This will install directories needed

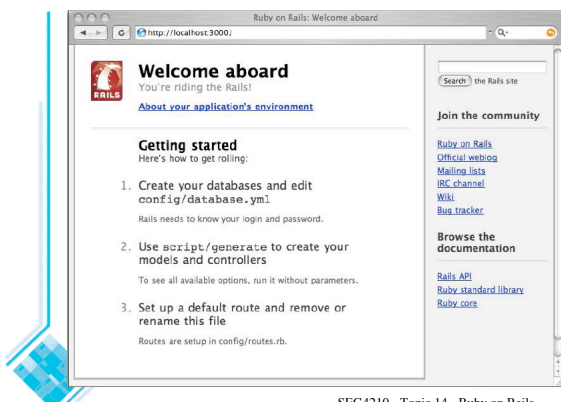
README components/ doc/ public/ tmp/
Rakefile config/ lib/ script/ vendor/
app/ db/ log/ test/

app contains controllers/, models/ and views/

Start the built-in server

`ruby script/server`

Here's what the default application looks like



Next step: Add a controller

`ruby script/generate controller Airline`

This will generate an empty (hook, or stub) class

```
• airline_controller.rb
class AirlineController < ApplicationController
end
```

Now let us add a method addflight to the Airline class

```
def addflight
end
```

We would call this from a URL as follows

`http://www.mysite.ca/airline/addflight`

Next step: Add a view using the Erb/rhtml approach

We need a 'template' file called `addflight.rhtml`

```
<html>
<head>
  <title>Add a flight</title>
</head>
<body>
  <h1>Add a flight</h1>
  <%= AirlineView.addFlightForm %>
</body>
</html>
```

Some things you can embed in rhtml code

Create an href to another ruby action

```
<%= link_to "Airline help", :action => "help" %>
```

We need to actually build a database to go any further

It is possible that the database is already there

- But Ruby provides nice facilities for setting up a database incrementally in Ruby code

First create an empty database using your database package

- MySQL is the easiest

Next test that Ruby can connect

```
rake db:migrate
```

Creating a simple database

The rule is one table per class

```
ruby script/generate model regularflight
```

You will then find a file

```
db/migrate/file 001_create_regularflights.rb
```

```
class CreateRegularflights < ActiveRecord::Migration
  def self.up
    create_table :regularFlights do |t|
      t.column :number, :integer
      t.column :departuretime, :string
      t.column :origin, :string
      t.column :destination, :string
    end
  end
end
```

Next steps

Make Ruby upgrade the database

```
rake db:migrate
```

Now create a controller admin

```
ruby script/generate controller admin
```

And edit `app/controllers/admin_controller.rb`

```
class AdminController < ApplicationController
  scaffold :regularflight
end
```

Go to <http://www.mysite.ca/airline/admin>

And you will see a page where you can add flights

Making Ruby upgrade the database: Adding a new column

```
class AddPrice < ActiveRecord::Migration
  def self.up
    add_column :products, :price, :decimal,
      :precision => 8, :scale => 2, :default => 0
  end

  def self.down
    remove_column :products, :price
  end
end
```

Ruby on Rails' naming

If you have a table called People, RoR will generate a class called Person and vice-versa

Other transformations:

- Specific_flights <-> SpecificFlight
- Person_roles <-> PersonRole

Enhancing model classes

When we create a new class/table the model code for the class is empty:

```
class Product < ActiveRecord::Base
end
```

Validation

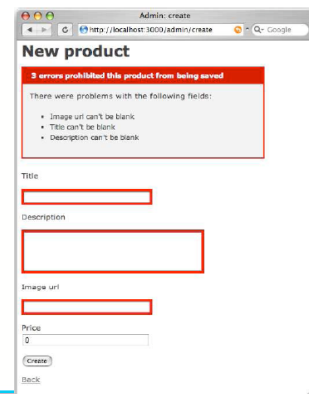
If we want to force certain data to be present

```
class Product < ActiveRecord::Base
  validates_presence_of :title, :description, :image_url
end
```

The generated web page for adding products will automatically ensure this data is filled in

- Fields with errors are highlighted
- Errors are summarized at the top of the form

Example generated web page



Other validations

```
validates_numericality_of :price
```

```
validates_uniqueness_of :title
```

```
def validate
  errors.add(:price, "should be at least 0.01" ) if
    price.nil? || price < 0.01
end
```

```
validates_format_of :image_url,
  :with => %r{\.(gif|jpg|png)$}i,
  :message => "must be a URL for a GIF, JPG, or PNG
image"
```

Class methods in the model to return a set of instances

```
class Product < ActiveRecord::Base
  # The self. Means it is a class (static) method
  def self.find_products_for_sale
    find(:all, :order => "title" )
  end
  # validation stuff...
end
```

Specifying associations in model classes

```
class Order < ActiveRecord::Base
  has_many :line_items
  ...
end

class Product < ActiveRecord::Base
  has_many :line_items
  has_many :orders, :through => :line_items
  ...
end

class LineItem < ActiveRecord::Base
  belongs_to :order
  belongs_to :product
  ...
end
```

SEG4210 - Topic 14 - Ruby on Rails

43

Generating automatic documentation

```
rake doc:app
```

```
rake stats
```

```
(in /Users/dave/Work/depot)
```

```
+-----+
|Name|Lines|LOC|Classes|Methods|M/C|LOC/M|
+-----+
|Helpers|17|15|0|1|10|13|
|Controllers|229|154|5|23|4|14|
|Components|0|0|0|0|0|0|
|Functional tests|206|141|8|25|13|3|
|Models|261|130|6|18|3|5|
|Unit tests|178|120|5|13|12|7|
|Libraries|0|0|0|0|0|0|
|Integration tests|192|130|2|10|5|11|
+-----+
|Total|1083|690|26|90|13|5|
+-----+
```

```
Code LOC: 299 Test LOC: 391 Code to Test Ratio: 1:1.3
```

SEG4210 - Topic 14 - Ruby on Rails

44

Automatic testing in RoR

Unit tests - test the model

- In test/unit

Functional tests - test a controller action

- In test/functional

Integration tests - test flow through the system

- In test/integration

SEG4210 - Topic 14 - Ruby on Rails

45

Sample initially generated test for Product

```
require File.dirname(__FILE__) + '/../test_helper'
class ProductTest < Test::Unit::TestCase
  #load test data
  fixtures :products
  def test_truth
    assert true
  end
end
```

SEG4210 - Topic 14 - Ruby on Rails

46

A test to verify that an empty product is not valid and that empty variables are invalid

```
def test_invalid_with_empty_attributes
  product = Product.new
  assert !product.valid?
  assert product.errors.invalid?(:title)
  assert product.errors.invalid?(:description)
  assert product.errors.invalid?(:price)
  assert product.errors.invalid?(:image_url)
end
```

SEG4210 - Topic 14 - Ruby on Rails

47

Checking some other properties

```
def test_positive_price
  product = Product.new(:title => "My Book Title" ,
    :description => "yyy" ,
    :image_url => "zzz.jpg" )
  product.price = -1
  assert !product.valid?
  assert_equal "should be at least 0.01" , product.errors.on(:price)

  product.price = 0
  assert !product.valid?
  assert_equal "should be at least 0.01" , product.errors.on(:price)

  product.price = 1
  assert product.valid?
end
```

SEG4210 - Topic 14 - Ruby on Rails

48

Topic 17

ORM: Object Relational Mapping

Rules for Mapping an Object Model to a Relational Database 1

General Rules

- A. Create one table for each ordinary class
 1. Make each instance a row in the table
 2. Make each simple attribute or qualifier into a column
 3. If a complex attribute is naturally composed of separate fields, make these into individual columns
 4. If a complex attribute consumes much space but is infrequently accessed, store it in a separate table.
 5. Make 1-1 or many-1 association into either
 - i. a column (using a foreign key)
 - ii. a separate table with foreign keys indicating each end of the relation

Object-Relational Mapping Rules 2

General rules cont ...

A. cont ...

6. Decide on the key:
 - The key must uniquely distinguish each row in the table
 - i. If unique, one or more of the attributes may constitute the key
 - ii. Otherwise, generate a column that will contain a special object-id as the key

Object-Relational Mapping Rules 3

General rules cont ...

- B. Create one table for each association that has many-many multiplicity (and for each association class)
 1. Make the keys of the tables being related to be foreign keys in this new table
 2. If we are creating an association class with additional associations connecting to it, then create a special object-id as the key
 3. Follow rules A1 to A5

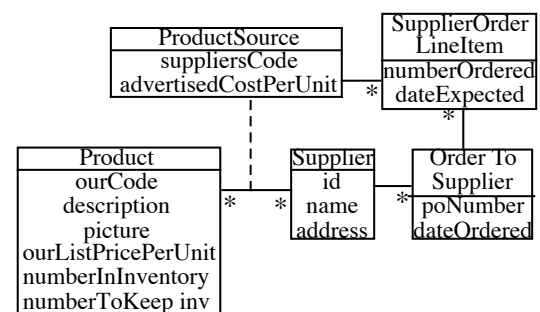
Object-Relational Mapping Rules 4

General rules cont ...

C. Deal with inheritance in either of the following two ways:

1. Keep inherited material in a common table.
 - For each object, create a row in both a table for the superclass and a table for the appropriate subclass
 - Relate the two tables using a foreign key in the subclass table
 - This is the recommended approach, but has the following drawbacks:
 - » It requires a join to access any object
 - » It requires inventing an identifier
2. Inherited columns can be duplicated in all subclass tables
 - There may be no need for a table representing the superclass
 - The duplication reduces maintainability

Example class diagram for an inventory system



Inventory System Tables 1

Resulting tables: Reason

Product	A
• product-code (key)	A2, A6i
• description	A2
• list-price-per-unit	A2
• number-in-inventory	A2
• number-to-keep-in-inv	A2

Product-Picture	A4
• product-code (foreign-key)	
• picture-bitmap	

Inventory System Tables 2

Supplier	A
• supplier-id (key)	A2, A6i
• name	A2
• street	A3
• city	A3
• postal-code	A3

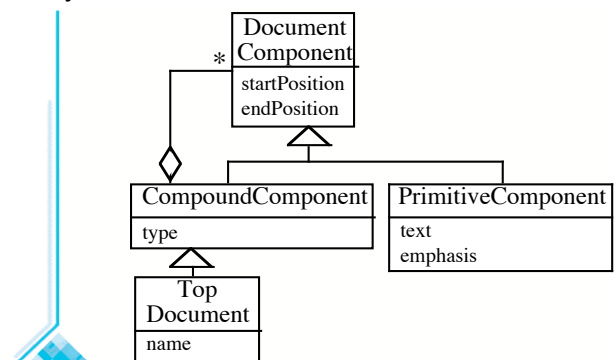
Product-Source	B
• product-source-id (key)	B2
• product-code (foreign-key)	B1
• supplier-id (foreign-key)	B1
• suppliers-code-for-product	A2
• advertised-cost-per-unit	A2

Inventory System Tables 3

Order-To-Supplier	A
• po-number (key)	A2, A6i
• supplier-id (foreign-key)	A5i
• date-ordered	A2

Supplier-Order-Line-Item	A
• line-item-id	A6ii
• po-number (foreign-key)	A5i
• product-source-id (foreign-key)	A5i
• number-ordered	A2
• date-expected	A2

Example Class Diagram for Document System - with Inheritance



Document System Tables 1

Resulting tables: Reason

Document-Component	A
• component-key (key)	A6ii
• start-pos	A2
• end-pos	A2

Compound-Component	A
• component-key (foreign-key)	C1
• type	A2

Document System Tables 2

Primitive-Component	A
• component-key (foreign-key)	C1
• text	A2
• emphasis	A2

Top-Document	A
• component-key (foreign-key)	C2
• type	C2
• name	A2

Part-Relation	A5ii
• part-key (foreign-key)	
• whole-key (foreign-key)	

Data modeling vs. OO modeling

Data Modelling followed by RDBMS use	Object-Oriented Modelling followed by OODMBS use
Associations computed using joins - Keys have to be developed	Associations are explicit using pointers
Only primitive data stored in columns (characters, numbers)	Structured data of arbitrary complexity can be stored.
Code independently developed and in different places	Code found in classes
An 'object' can be distributed among tables - A problem for complex objects	An object in one place, so is fast to access
Associations are by default bi-directional - I am the son of my father and he is the father of me	Associations can be unidirectional - I know the Queen, but she doesn't know me

SEG4210 – Advanced Software Design and Reengineering

TOPIC 13 Ajax

What is Ajax? A *style* of web app!

- Asynchronous
 - Bits of data downloaded when needed
 - Download is initiated, but may not complete for a while
 - Meanwhile the user can continue working
- Javascript
 - The language used in Web Browsers to manipulate what appears
- With XML
 - The format of the data downloaded the JavaScript to modify what appears on the page

AJAX permits Rich Internet Applications (RIA)

Applications that look and feel like desktop apps

- In whole or in part

A key part of “Web 2.0”

Suggested reference books for AJAX

Visual Quickstart Guide: JavaScript and Ajax, 6th edition

- By Tom Negrino and Dori Smith, 2007, Peachpit Press
- Website: <http://www.javascriptworld.com/>
 - Sample code in these slides came from here
- Covers css and everything you need to know

Ajax Hacks: Tips and Tools for Creating Responsive Web Sites

- Bruce W. Perry, 2006 O'Reilly
- Also covers basics of Ruby on Rails

The Javascript Language

Originally to be called LiveScript

- Developed by Netscape

Relationship to Java?

- Not directly, but ...
 - Shares syntax, keywords
 - Named to take advantage of Java mania

Variants

- Microsoft developed its own JScript (mostly the same)
- A common subset of both is standardized as ECMAScript

Key things JavaScript/Ajax is useful for 1

Auto-completing forms

Forms that have data dependencies

- Ask different questions of a Canadian, American etc.

Validating forms before they are submitted to the server

- Error messages can be popped up right under the appropriate field
- With Ajax you can obtain data to help in the validation
 - e.g. fill in the address of a postal code)

Working with Dates

- Displaying calendars in a date field
- With Ajax, you can fill in details from the selected date

Key things JavaScript/Ajax is useful for 2

Menu and animation effects not available in basic html

- But don't overdo fancy effects
- People need to intuitively understand how something will work

Helping create mashups

- Incorporating data from other sites
- Adding Google maps is the most famous example
 - www.google.com/apis/maps/documentation/
- You can also do 'scraping' of data from other sides
 - But this requires server-side help

Key things JavaScript/Ajax is useful for 3

Storing data semi-persistently in cookies

- Allowing data to be accumulated or to persist from page to page or session to session without having to store it on the server
- But don't rely on the data to be saved
 - Users are free to trash their cookies
- Examples of use
 - A multi-page questionnaire that only submits the answers at the very end.
 - Tracking settings on pages in the site you have visited so they appear the same next time

Key benefits/attractions of AJAX style

Whole pages don't reload, leading to

- Smooth response
- The user doesn't feel so lost
 - they in the same place until logically ready to go somewhere else
- Ability to hide, show information on demand to fill in bits of the window

Consider using Ajax packages 1

- Yahoo User Interface Library
 - <http://developer.yahoo.com/yui/>
 - sourceforge.net/projects/yui/
- Script.aculo.us
 - <http://Script.aculo.us>
- Google web toolkit
 - Write code in Java that is compiled to Javascript
 - <http://code.google.com/webtoolkit/>
- Dojo
 - <http://www.dojotoolkit.org/>
- jQuery
 - <http://jquery.com/>

Consider using Ajax packages 2

- Moo.fx
 - <http://moofx.mad4milk.net/>
- Open Rico
 - <http://openrico.org/>
- Tibco General Interface
 - <http://www.tibco.com/devnet/gi/default.jsp>

What JavaScript (and Ajax) can not do for security reasons

Read or write

- A file on the client (browser) computer
 - Except 'cookies'

Write

- Directly to a server's file
 - must invoke a program there

Read

- Contents of a displayed web page that came from another server
- Data from another server
 - You have to program the server to obtain the data that the client will need

General guidelines for Javascript/Ajax programming 1

Where possible only the ECMAScript (3rd edition) subset of Javascript

Make sure everything works in IE6, 7 and Firefox

- Otherwise you will anger a large number of people
- In a few cases you will have to program 'if' statements
- Test, test, test ...

Decide on whether you will support pre-IE4, pre NETSCAPE6 browsers

- If not, make sure that those browser users at least get a warning, not just bizarre behaviour

General guidelines for Javascript/Ajax programming 2

Create the web page in basic good-quality html first

- Will have reduced functionality, but will still provide the basic information
- Important for the disabled or those with Javascript off
- Then override the normal functionality of links to add Ajax features
 - Sometimes this is called 'Hijax'

Don't be evil

- Avoid popup advertising or, in fact, any popups
- Ensure the back button keeps working
- Let people bookmark pages

General guidelines for Javascript/Ajax programming 3

Separate concerns

- Basic content in a set of .html file
- Code in a set of .js files
- Format information in a set of .css files
- Downloadable data for your application in .xml or .json files
- All the above may be generated
 - By php, asp, java etc.

Follow good UI design guidelines

- See useit.com

General guidelines for Javascript/Ajax programming 4

Use frames and manipulation of windows sparingly

- Consider iframes rather than regular frames

Use plain html unless the benefits of JavaScript outweigh the costs

Key technologies you need to understand to do Ajax programming

JavaScript itself

XHTML

- The latest version of html that conforms to XML

Document Object Model (DOM)

- The tree of objects inside a web page

XML

XMLHttpRequest

- The function JavaScript uses to obtain data from the server

Cascading Style Sheets

- Separately describe fonts, colours and positioning from html

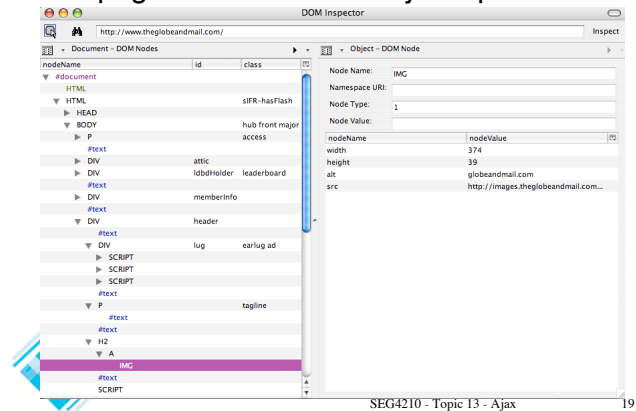
XHTML - key points

All tags should be opened and closed

- Old style: <p> at the end of a paragraph.
- New style: <p> at the start, </p> at the end

Use <div> or perhaps to give sections of the web page classes and ideas

The DOM - Describes the structure of a webpage in terms of the objects present



Key points about the DOM

CSS can set the style and position any object that has an ID or a class

JavaScript can query or modify any object

- Specific functions operate on each object type

JavaScript can add new objects and delete them at any level

Event handling as the main way things are triggered in an Ajax application

Working with mouse clicks

- onclick, ondblclick
- onmousedown, onmouseup

Working with mouse movement

- onmousemove, onmouseover, onmouseout

Working with DOM object selected as the current UI object

- onfocus, onblur

Working with arbitrary keys being pressed

- onkeypress, onkeydown, onkeyup

Dealing with window changes

- onresize

More on events

The events on the previous page work with Many DOM objects

- Document, Applet, Area, Body, Link

DOM objects for form elements also accept event 'onchange'

- Button, Checkbox, Password, Radio, Text, Textarea

Events 'onsubmit', 'onreset' also accepted by

- Form

Events 'onload', 'onunload' accepted by

- Window

Getting started with JavaScript

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>My second script</title>
  <script language="Javascript" type="text/javascript"
    src="script02.js">
  </script>
</head>
<body bgcolor="#FFFFFF">
  <h1 id="helloMessage">
  </h1>
</body>
</html>
```

Code for script02.js

```
window.onload = writeMessage;

function writeMessage() {

document.getElementById("helloMessage").innerHTML =
"Hello, world!";
}
```

Overriding a link

In the html

- Welcome to our site... c'mon in!

The javascript

```
window.onload = initAll;

function initAll() {
    document.getElementById("redirect").onclick = initRedirect;
}

function initRedirect() {
    window.location = "jswelcome.html";
    return false;
}
```

A trivial AJAX application - html code

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN">
<html>
<head>
    <title>My First Ajax Script</title>
    <script src="script01.js" type="text/javascript"
        language="Javascript">
    </script>
</head>
<body>
    <p><a id="makeTextRequest" href="gAddress.txt">Request a
        text file</a><br />
    <a id="makeXMLRequest" href="us-states.xml">Request an
        XML file</a></p>
    <div id="updateArea">&nbsp;</div>
</body>
</html>
```

Ajax Java code for script01.js - part 1

```
window.onload = initAll;
var xhr = false;

function initAll() {
    document.getElementById("makeTextRequest").onclick =
        getNewFile;
    document.getElementById("makeXMLRequest").onclick
    =
        getNewFile;
}

function getNewFile() {
    makeRequest(this.href);
    return false;
}
```

Ajax Java code for script01.js - part 2

```
function makeRequest(url) {
    if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest(); // for non-IE browsers
    }
    else { // for IE 5 and 6 browsers
        if (window.ActiveXObject) {
            try {
                xhr = new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch (e) {}
        }
    }

    if (xhr) {
        xhr.onreadystatechange = showContents; // what should happen when request done
        xhr.open("GET", url, true); // third argument says the request is synchronous
        xhr.send(null); // actually sends off the request
    }
    else { // browser is too old
        document.getElementById("updateArea").innerHTML = "Sorry, but I couldn't create an
XMLHttpRequest";
    }
}
```

Ajax Java code for script01.js - part 3

```
function showContents() {
    if (xhr.readyState == 4) {
        if (xhr.status == 200) {
            var outMsg = (
                xhr.responseXML &&

                xhr.responseXML
                .getElementsByTagName(
                    "choices")[0].textContent :
                xhr.responseText;

            }
            else {
                var outMsg = "There was a problem with the request "
                +
                xhr.status;
            }
            document.getElementById("updateArea").innerHTML = outMsg;
        }
    }
}
```

Cascading Style Sheets - Quick Overview

To specify that an html file will use a certain stylesheet:

In the <head> section, put:

```
<style type="text/css">@import "mysheet.css";</style>
```

CSS example 1

Sample css cod

```
body {  
    background-color: white;  
    color: black;  
    font-weight: bold;  
}  
div {  
    font-weight: normal;  
}  
img {  
    margin-right: 10px;  
}
```

CSS example 2

```
body {  
    background-color: white;  
    color: black;  
    font-family: "Trebuchet MS", verdana, helvetica, arial, sans-serif;  
    font-size: 0.9em;  
}  
img {  
    margin-right: 10px;  
}  
.character {  
    font-weight: bold;  
    text-transform: uppercase;  
}  
div.character {  
    margin: 1.5em 0em 1em 17em;  
}  
.directions {  
    font-style: italic;  
}  
div.dialog, .directions {  
    margin-left: 22em;  
}
```

CSS example 3 - for a particular ID

```
#week 2 {  
    background-color : #FFFF00;  
}
```

TOPIC 18

Enterprise JavaBeans (EJB) and other Component Frameworks (CORBA, Spring Framework, Guice)

Software Components

- Component-based development aims at enabling a higher level of software reuse
- A software component
 - is a package of software that provides some functionality
 - has well defined services
 - can be used in building many different applications
 - can be reused in building larger components
- Many allow cross-language and cross-platform reuse

What are JavaBeans?

- Java Beans are an early Java (1996) mechanism for building software components
- “A Bean is a reusable software component that can be manipulated visually using a builder tool” (Sun Java Doc.)
 - It appears to the programmer as just a class that follows a set of conventions
 - The class may internally reference other classes
- The JavaBeans API allows creating reusable, platform-independent components

Introspection

- Introspection is a process that is used by builder tools to discover a Bean's features such as
 - properties, methods, and events
- Beans support introspection in two ways:
 1. By adhering to specific conventions, when naming Bean features
 - the Introspector class examines Beans for these conventions to discover Bean features
 - The Introspector class relies on the core reflection API

Introspection (Cont.)

2. By explicitly providing property, method, and event information with a related Bean Information class
 - A Bean information class implements the BeanInfo interface
 - A BeanInfo class lists the Bean features that are explored by builder tools

Bean minimal conventions

Have a no-argument constructor

Properties are named getX, setX, isX

Class must be serializable

- Often internally a Bean object contains other objects
- The whole Bean is serialized

Example Minimal Bean

```
// PersonBean.java
public class PersonBean implements java.io.Serializable {
    private String name;
    private boolean deceased;

    // No-arg constructor (takes no arguments).
    public PersonBean() {}

    public String getName() {
        return this.name;
    }
    public void setName(String name) {
        this.name = name;
    }

    // Different semantics for a boolean field (is vs. get)
    public boolean isDeceased() {
        return this.deceased;
    }
    public void setDeceased(boolean deceased) {
        this.deceased = deceased;
    }
}
```

SEG4210 - Topic 18 - Component Frameworks

7

Tools to work with JavaBeans

Eclipse visual editor
NetBeans GUI Editor

Many Java libraries follow bean convention

- AWT
- Swing
- SWT

SEG4210 - Topic 18 - Component Frameworks

8

Event Model

- Events provides a notification mechanism between a source object and one or more listener objects
- Beans use events to communicate with each other.
- A Bean that wants to receive events (a listener Bean) registers its interest with the Bean that sends the event (a source Bean)
- Builder tools can examine a Bean and determine which events that Bean can send and which it can handle receive

SEG4210 - Topic 18 - Component Frameworks

9

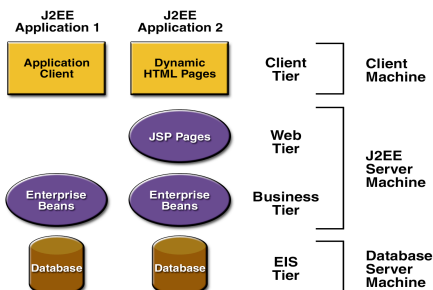
Java 2 Enterprise Edition (J2EE)

- The J2EE is used to develop multi-tiered distributed applications
- J2EE includes the following tiers (next slide):
 - Client-tier components run on the client machine
 - Web-tier components (optional) run on the J2EE server
 - Business-tier components run on the J2EE server
 - Enterprise information system (EIS)-tier software runs on the EIS server
- Application logic is divided into components and each component is usually installed on a different machine
 - according to the tier it belongs to

SEG4210 - Topic 18 - Component Frameworks

10

J2EE Tiers



Source: Sun Java Doc.

SEG4210 - Topic 18 - Component Frameworks

11

J2EE Components

- The J2EE specification defines the following components:
 - Application clients and applets run on the client
 - Java Servlet and JavaServer Pages (JSP) are Web components that run on the server
 - *Enterprise JavaBeans* (EJB) components represent business components and run on the server side
- J2EE components are written in Java in the same way ordinary Java programs are created
- All J2EE components come in entities called *containers*
 - Containers provide components with services such as life cycle management, security, deployment, and threading

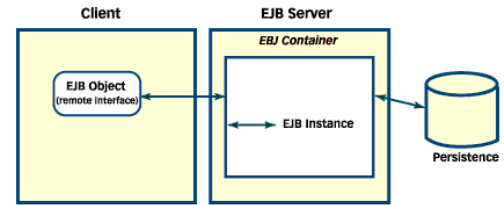
SEG4210 - Topic 18 - Component Frameworks

12

Enterprise JavaBeans (EJB)

- <http://java.sun.com/products/ejb/>
- Not actually a type of JavaBeans !!
 - The only similarity is that they are components
- Runs on the server side
 - Persistence through database storage
 - Incorporate transaction processing, concurrency control
- Represents an element of business logic
- A business logic is the code that represents the solution to a business problem
 - E.g. Class diagram of an inventory system

Basic EJB architecture



- The EJB object on the client side is generated.
 - Client code communicates with the real object on the server
 - Both client and server side objects have the same Java interface

How an EJB application works

Client program contacts the container

- Requests that a particular type of EJB be created or found
 - Uses Java Naming and Directory Interface (JNDI) to locate the 'home interface' for the Bean
 - Calls the home interface
- A 'remote' object appears on the client side in response

Benefits of Enterprise JavaBeans 1

- Simplify the development of large, distributed applications
- The EJB container provides transaction management, security authorization, etc.
 - The bean developer can concentrate on solving business problems
- The beans contain the application's business logic,
 - The client developer can focus on the presentation of the client

Benefits of Enterprise JavaBeans 2

- The client developer does not have to code the routines that implement business rules or access databases
 - Clients are thinner
 - important for clients that run on small devices
- Enterprise beans are portable components,
 - One can build new applications from existing beans

Types of Enterprise Beans

- **Entity Beans:**
 - Represents persistent data in a database
 - Has methods that act on that data
- **Session Beans:**
 - Created by a client
 - Exist only for the duration of a single session
 - Perform operations on behalf of the client such as reading, writing, or updating a database;
 - Do not represent data that is stored in a database.
 - Can be stateless or stateful
- **Message Driven Beans**
 - Used for asynchronous requests
 - Do not require a response (inform me of any update)

Overview of CORBA

- Common Object Request Broker Architecture
- It is a industry standard, language neutral, platform independent, open system for doing distributed computing
- Maintained by the Object Management Group (OMG)
 - Is considered a little ‘old’, since it has been around since 1991
 - But many more recent technologies have borrowed the same ideas
- Widely used in industries such as telecommunications, health care, and banking...

CORBA Characteristics

- Platform Neutral:
 - CORBA is designed to specify how systems should communicate in an heterogeneous environment
 - Different OS, programming languages and architectures
 - Currently there are mappings for C, C++, Java, COBOL, Ruby, Smalltalk and other languages
- Open:
 - OMG specifications are open to anyone who wishes to implement them

Object Request Broker

- Communications over CORBA are based on the client-server principle
 - The client sends a request to the server, which returns a response
 - Who is the client and who is the server is defined by the request
- All requests are sent through interfaces and go through the *Object Request Broker (ORB)*
- The ORB acts as an intermediary,
 - taking client requests
 - formatting them in a manner to send over TCP/IP based connections
 - decoding them on the other end and
 - delivering the request to the server
- The work done by ORB is transparent to the clients and servers

CORBA Interfaces

- Interfaces define the services that a client is allowed to call upon a server without specifying any implementation.
- A CORBA object is said to support or implement an interface
- Interfaces are specified in CORBA's Interface Definition Language (IDL)
- IDL is used to define the semantics of the method calls
 - This includes the types and number of arguments and the return type
- CORBA methods can also raise or throw exceptions to signal error conditions

Inversion of Control 1

Also known as ‘dependency injection’

- A concept used in several frameworks we will briefly discuss

The basic idea of a dependency

- If class X has an association to class Y and calls a method of Y
 - then X depends on Y
- In many older frameworks the dependency would be created by code like this:

```
class X {
    Y aY;
    ...
    aY.getY()
}
```

- This means that the code of class X must know about class Y
- And class X controls the nature of the dependency

Inversion of Control 2

To invert control, class Y creates the X and sets of the dependency to a Y

- Through a constructor argument
- Or, through a setter method
- Or, using a factory method

Result: Looser coupling

- Easier to plug in new components (Y can use something other than X)
- Easier to test

Spring Framework

See

- www.springframework.org

Developed by Rod Johnson in 2000 to 2004 timeframe

- Has become popular since then

An alternative to EJB

- Also runs under Java
- Uses JavaBeans
- Has as objectives to be simpler and more consistent than EJB

Spring Framework Key Contributions 1

Inversion of control container

- Bean Factory
- Objects created are called *managed objects* and are JavaBeans
- Configured by loading XML files that contain Bean definitions
- Dependency lookup: Caller asks the container object for an object with a specific name or type

Integration with multiple different persistence/data-access frameworks

- JDBC, iBatis, Hibernate, and others

MVC web application framework

Has its own aspect-oriented framework

Spring Framework Key Contributions 2

Transaction management

- Can work with
 - Local transactions
 - that don't require a server
 - Nested transactions

One more interesting up-and-coming framework: Guice

Google's dependency-injection framework for creating components

- <http://code.google.com/p/google-guice/>

Key contributions:

- Very small and fast
- Reduces amount of code needed to create code with low coupling
- Takes advantage of new features of Java
 - Annotations and generics

Java annotations

Metadata accessible to the application at runtime using reflection

- See <http://www.developer.com/java/other/article.php/3556176>

Guice example

- `@Retention(RUNTIME)` // visible at runtime
- `@Target({ FIELD, PARAMETER })`
- `@BindingAnnotation`
- `public @interface Named {`
- `String value();`

SEG4210
Advanced Software Design and Reengineering

TOPIC 16
Software Maintenance, Evolution, Program Comprehension,
and Reverse Engineering

Software Maintenance

- The modification of a software product after delivery
 - to correct faults
 - to improve performance or other attributes
 - or to adapt the product to a changed environment

ANSI/IEEE Standard 729-1983

Software Maintenance Types 1

Corrective maintenance:

- Fixing faults that cause the system to fail

Adaptive maintenance:

- Making changes in existing software to accommodate a changing environment

Enhancement

- Adding new features

Software Maintenance Types 2

Perfective maintenance

- Making improvements to the existing system without effecting end-user functionality
 - To make it easier to extend and add new features in the future
 - Also called *re-engineering*
 - *Refactoring* is a one type of perfective maintenance

Preventive maintenance

- Preventing failures by fixing defects in advance of failures
- A kind of perfective maintenance
- Key examples: Y2K and Daylight Savings Time adjustments

Software Maintenance Steps 1

Understand the existing system

- Study whatever form of documentation exists about the system to be modified
 - Often the only reliable source of information is the source code
- Use tools to recover the high-level design models of the system

Software Maintenance Steps 2

Define the maintenance objectives

- Set the requirements

Analysis

- Evaluate alternatives for handling the modification
 - Estimate the costs and benefits of the alternative modifications
 - Perform *impact analysis*
 - Determine the effect of the change on the rest of the system

Software Maintenance Steps 3

Design, implement, and test the changes

Revalidate

- Running *regression tests* to make sure that the unchanged code still works and is not adversely affected by the new changes

Software Maintenance Steps 4

Train

- Inform users of the changes

Convert and release

- Generate and release/install a new version with the modified parts
- May involve *migrating* database schema changes and data at the same time

Software Evolution 1

Evolution: a term that is now often preferred as a replacement for 'maintenance'

Lehman's 8 Laws of Software Evolution

1. **Continuing Change:** A system must be continually adapted else it becomes progressively less satisfactory in use
2. **Increasing Complexity:** As a system is evolved its complexity increases unless work is done to maintain or reduce it
3. **Self Regulation:** Global system evolution processes are self-regulating
4. **Conservation of Organizational Stability:** Average activity rate in an evolution process tends to remain constant over system lifetime or segments of that lifetime

Software Evolution 2

Lehman's 8 laws continued

5. **Conservation of Familiarity:** In general, the average incremental growth, or growth rate trend, tends to decline
6. **Continuing Growth:** The functional capability of systems must be continually increased to maintain user satisfaction over the system lifetime
7. **Declining Quality:** Unless rigorously adapted to take into account changes in the operational environment, the quality of a system will appear to decline as it is evolved
8. **Feedback System:** Evolution processes are multi-level, multi-loop, multi-agent feedback systems

Program Comprehension

Program comprehension:

- The discipline concerned with studying the way software engineers understand programs

Objective of those studying program comprehension:

- design tools that will facilitate the understanding of large programs

Program Comprehension Strategies 1

The bottom-up model:

- Comprehension starts with the source code and abstracting from it to reach the overall comprehension of the system
- Steps:
 - Read the source code
 - Mentally group together low-level programming details (*chunks*) to build higher-level abstractions
 - Repeat until a high-level understanding of the program is formed

Program Comprehension Strategies 2

The top down model:

- Comprehension starts with a general idea, or hypothesis, about how the system works
 - Often obtained from a very quick look at what components exist
- Steps
 - First formulate hypotheses about the system functionality
 - Verify whether these hypotheses are valid or not
 - Create other hypotheses, forming a hierarchy of hypotheses
 - Continue until the low-level hypotheses are matched to the source code and proven to be valid or not

Program Comprehension Strategies 3

The Integrated Model:

- Combines the top down and bottom up approaches
- Empirical results show that maintainers tend to switch among the different comprehension strategies depending on
 - The code under investigation
 - Their expertise with the system

Partial Comprehension

Usually is not necessarily to understand the whole system if only part of it needs to be maintained

- But a high fraction of bugs arise from not understanding enough!

Most software maintenance tasks can be met by answering seven basic questions:

- How does *control flow* reach a particular location?
- Where is a particular subroutine or procedure *invoked*?
- What are the *arguments and results* of a function?
- Where is a particular *variable set, used or queried*?
- Where is a particular variable *declared*?
- What are the *input and output* of a particular module?
- Where are *data objects accessed*?

Reverse Engineering

- The process of analyzing a subject system
 - to identify the system's components and their inter-relationships
 - and to create representations of the system, in another form, at a higher level of abstraction"

Chikofsky and Cross

Two main levels of reverse engineering

Binary reverse engineering

- Take a binary executable
 - Recover source code you can then modify
- Useful for companies that have lost their source code
- Used extensively by hackers
- Can be used legally, e.g. to enable your system to interface to existing system
- Illegal in some contexts

Source code reverse engineering

- Take source code
 - Recover high level design information
- By far the most widely performed type of reverse engineering
- Binary reverse engineers also generally do this too

Reverse Engineering Objectives 1

Cope with complexity:

- Have a better understanding of voluminous and complex systems
- Extract relevant information and leave out low-level details

Generate alternative views:

- Enable the designers to analyze the system from different angles

Reverse Engineering Objectives 2

Recover lost information:

- Changes made to the system are often undocumented;
 - This enlarges the gap between the design and the implementation
- Reverse engineering techniques retrieve the lost information

Reverse Engineering Objectives 3

Detect side effects:

- Detect problems due to the effect a change may have on the system before it results in failure

Synthesize higher-level abstractions

Reverse Engineering Objectives 4

Facilitate reuse

- Detect candidate system components that can be reused

Source code reverse engineering techniques

Program analysis

- To be discussed in a separate module of this course

Program slicing

Design recovery

Architecture recovery

Program Slicing

A form of data flow analysis

- concerned with analyzing all the statements in a program that may
 - *Affect the value of a given variable at a certain point* during execution
 - » Looking backward
 - Be affected by the execution of a certain statement
 - » Looking forward

Can be either static or dynamic

Static slicing

- Considers all possible inputs
 - the resulting slices are usually quite large

Dynamic slicing

- Extracting parts of the program that contribute to the computation of the function according to a specific input

Design Recovery 1

Create design abstractions in order to understand what a program does, how it does it and why it does it

Examine multiple sources of knowledge including

- the system documentation (if available),
- the knowledge that the software engineers have of the system

Design Recovery 2

Difficult to perform on large systems that have undergone ad-hoc maintenance for a long period of time

- Documentation of such *legacy systems* is usually out of date
- Original developers are usually no longer working within the organization

Architecture Recovery

Aims to recover the overall system structure in terms of its high-level components and the way they interact

There are several techniques

- Using human experts
- Recognizing known patterns
- Static and dynamic analysis
- Clustering and data mining

Topic 15

Program Analysis and Transformation

Copyright Note

These slides are derived from work by
[Bil Tzerpos](#) a faculty member at York University

Program Analysis

Extracting information, in order to *present abstractions* of, or *answer questions* about, a software system

Static Analysis

- Examines the source code

Dynamic Analysis

- Examines the system as it is executing

What are we looking for when performing program analysis?

Depends on our goals and the system

- In almost any language, we can find out information about variable usage
- In an OO environment, we can find out which classes use other classes, what is the inheritance structure, etc.
- We can also find potential blocks of code that can never be executed in running the program (dead code)
- Typically, the information extracted is in terms of entities and relationships
 - Can be modelled in a class diagram

Entities

Entities are individuals that live in the system, and attributes associated with them.

Some examples:

- Classes, along with information about their superclass, their scope, and 'where' in the code they exist.
- Methods/functions and what their return type or parameter list is, etc.
- Variables and what their types are, and whether or not they are static, etc.

Relationships

Relationships are interactions between the entities in the system

Relationships include

- Classes inheriting from one another.
- Methods in one class calling the methods of another class, and methods within the same class calling one another.
- A method referencing an attribute.

Information format for data extracted during program analysis

Many different formats in use

- Simple but effective: RSF (Rigi Standard Format)
inherit TRIANGLE SHAPE
- TA (Tuple Attribute) is an extension of RSF that includes a schema
\$INSTANCE SHAPE Class
- GXL is a XML-based extension of TA
 - Blow-up factor of 10 or more makes it rather cumbersome
- New formats based on YAML being developed

Representation of extracted information

A fundamental issue in re-engineering

- Provides
 - means to generate abstractions
 - input to a computational model for analyzing and reasoning about programs
 - means for translation and normalization of programs

Key questions regarding representations of extracted information

What are the strengths and weaknesses of each representations of programs?

What levels of abstraction are useful?

Abstract Syntax Trees

A translation of the source text in terms of operands and operators

Omits superficial details, such as comments, whitespace

All necessary information to generate further abstractions is maintained

AST production

Four necessary elements to produce an AST:

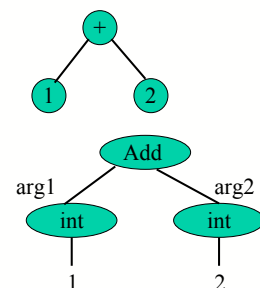
- Lexical analyzer (turn input strings into tokens)
- Grammar (turn tokens into a parse tree)
- Domain Model (defines the nodes and arcs allowable in the AST)
- Linker (annotates the AST with global information, e.g. data types, scoping etc.)

AST example

Input string: 1 + /* two */ 2

Parse Tree:

AST (without global info)



Static Analysis

Involves parsing the source code

Usually creates an *Abstract Syntax Tree*

Borrows heavily from compiler technology

- but stops before code generation

Requires a grammar for the programming language

Can be very difficult to get right

CppETS

CppETS is a benchmark for C++ extractors

A collection of C++ programs that pose various problems commonly found in parsing and reverse engineering

Static analysis research tools typically get about 60% of the problems right

Example program

```
#include <iostream.h>
class Hello {
public: Hello(); ~Hello();
};
Hello::Hello()
{ cout << "Hello, world.\n"; }

Hello::~~Hello()
{ cout << "Goodbye, cruel world.\n"; }

main() {
    Hello h;
    return 0;
}
```

Example Q&A

How many member methods are in the Hello class?

Answer: Two, the constructor (Hello::Hello()) and destructor (Hello::~~Hello())

Where are these member methods used?

Answer: The constructor is called implicitly when an instance of the class is created. The destructor is called implicitly when the execution leaves the scope of the instance.

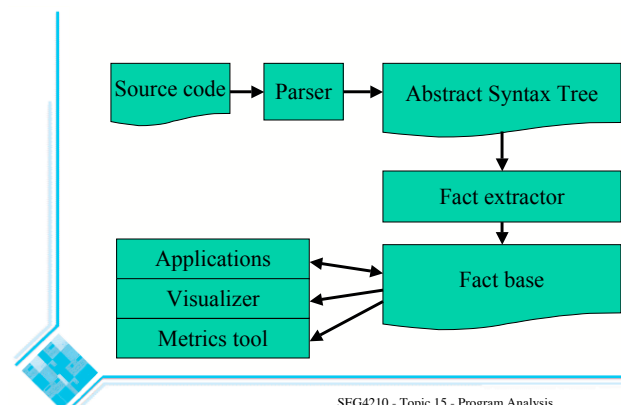
Static analysis in IDEs

High-level languages lend themselves better to static analysis needs

- Rational Software Modeler does this with UML and Java

Unfortunately, most legacy systems are not written in either of these languages

Static analysis pipeline



Dynamic Analysis

Provides information about the run-time behaviour of software systems, e.g.

- Component interactions
- Event traces
- Concurrent behaviour
- Code coverage
- Memory management

Can be done with a debugger

Instrumentation

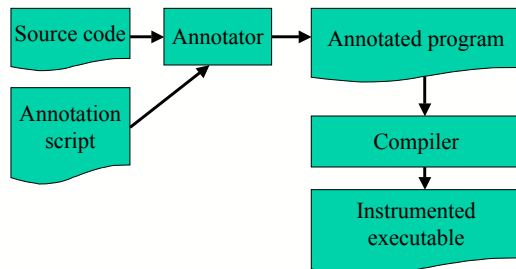
Augments the subject program with code that

- transmits events to a monitoring application
- or writes relevant information to an output file

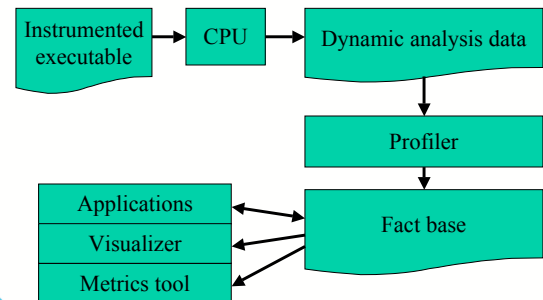
A *profiler* tool can be used to examine the output file and extract relevant facts from it

Instrumentation affects the execution speed and storage space requirements of the system

Instrumentation process



Dynamic analysis pipeline



Non-instrumented approach

One can also use debugger log files to obtain dynamic information

- Disadvantage: Limited amount of information provided
- Advantages: Less intrusive, more accurate performance measurements

Dynamic analysis issues

Ensuring good code coverage is a key concern

A comprehensive test suite is required to ensure that all paths in the code will be exercised

Results may not generalize to future executions

The size of run-time information is extraordinary large

Summary: Static vs. Dynamic Analysis

Static Analysis

Reasons over all possible behaviours (general results)

Conservative and sound

Challenge: Choose good abstractions

Dynamic Analysis

Observes a small number of behaviours (specific results)

Precise and fast

Challenge: Select representative test cases

Program Transformation

The act of changing one program into another

- from a source language to a target language

This is possible because of a program's well-defined structure

- But for validity, we have to be aware of the semantics of each structure

Used in many areas of software engineering:

- Compiler construction
- Software visualization
- Documentation generation
- Automatic software renovation

Program transformation application examples

Converting to a new language dialect

Migrating from a procedural language to an object-oriented one, e.g. C to C++

Adding code comments

Requirement upgrading

- e.g. using 4 digits for years instead of 2 (Y2K)

Structural improvements

- e.g. changing GOTOs to control structures

Pretty printing

Simple program transformation

Modify all arithmetic expressions to reduce the number of parentheses using the formula: $(a+b)*c = a*c + b*c$

$x := (2+5) * 3$

becomes

$x := 2*3 + 5*3$

Two types of transformations

Translation

- Source and target language are different
- Semantics remain the same

Rephrasing

- Source and target language are the same
- Goal is to improve some aspect of the program such as its understandability or performance
- Semantics might change

Transformation tools

There are many transformation tools

Program-Transformation.org lists 90 of them

- <http://www.program-transformation.org/>
- TXL is one of the best

Most are based on 'term rewriting'

- Other solutions use functional programming, lambda calculus, etc.

SEG 4210 – Advanced Software Design and Reengineering

Topic 12 Introduction to Refactoring

Copyright Note

Based on the book:

“Refactoring, Improving the Design of Existing Code”

- Martin Fowler
- 2000, Addison-Wesley
- ISBN 0-201-48567-2

What is Refactoring?

“Refactoring is the process of *changing* a software system in such a way that it *does not alter the external behaviour* of the code yet improves its internal structure.”

What is Refactoring? (cont.)

Each refactoring is a series of steps

Each step is simple and small

- Minimises introduction of new bugs

Unit tests should be used to verify changes have no effect on behaviour

Behavioural changes to a program are not refactorings

Why Refactor?

It improves design so that maintenance is easier

- Makes source code easy to read and understand
 - I.e. makes the code more usable by programmers
- Reduced chaos in long-lived code
- Faster programming in the long term

Thinking about specific ways to refactor helps us formalize and name best practices

Opportunities for Refactoring

During initial design and programming

When adding functionality

- Make adding new feature easier
- Keep refactoring separate, though

When fixing a bug

During a code review

- E.g. in pair programming

Specific Refactorings to be Discussed in the Following Slides

Many refactorings make use of other simpler refactorings

Some are opposites (inverses)

- Use the one that makes your code easier to understand

Some can be applied straightforwardly whereas others require deeper analysis

Using a tool to help in refactoring is useful

- Many require finding code to change elsewhere, e.g. code that has to change to be consistent with the refactoring

Refactoring: *Extract Method*

Probably the most used refactoring

- You should have done this many times yourself

Identify code that should be put into a separate method that you will then call

- Code to reuse
 - Immediately, or potentially
- Code that is making a method too long,
- Code that is difficult to understand
 - The new method's name helps explain the function

You may also do a *Move Method* refactoring too move the method to a different class

Dealing with Variables when doing the *Extract Method* Refactoring

Local variables only used in the extracted code can be copied to the new method

If one original variable is modified, then

- Make the new method a function

If several original variables are modified, then do one of the following

- Pass *references* (usually not a good idea since adds complexity)
- Use *output* or *var parameters* available in some languages
- Structure the variable to modify as an *object*, then update the fields of the object
- Use other refactorings first so multiple variables do not need to be modified

Inverse Refactoring: *Inline method*

Much less common than Extract Method, but still useful sometimes

- For efficiency
- If the inlined method is only called once, and overall complexity would be less after the inlining

Refactoring: *Extract Class*

Performed when a you perform object-oriented analysis and the resulting UML class diagram suggests *two classes* where you have *only one*

- E.g. Extract a 'common superclass'
- E.g. Extract a class that will be associated using the 'Player Role' or 'Abstraction Occurrence' patterns
- The need results from a violation of the principle of *separation of concerns*
 - Often the result of feature creep
 - Can result from too many *Extract Method* refactorings

Split out related subset of data and methods into a new class

May need to rename the old class if its duties have changed

Extract Class - 2

Normally you need a reference to an object of the new class in the old one

Uses *Move Field* and *Move Method* refactorings to move elements of the new class across (test after each move)

Inverse Refactoring: *Inline Class*

The opposite of *Extract Class*

Applied when a class is doing too little

- Normally only when there is a 1-1 association

Move fields and methods into a class that uses them

Refactoring: *Inline Temp*

Replace a local variable with a simple expression

- Can make other refactorings easier
- Can improve efficiency slightly

Only do it if the variable is not being used in many places

Local variable may be doing no harm, can be making the code easier to understand, or may be there to avoid breaking the *Law of Demeter*

- In which case, leave it there and don't do this refactoring

Refactoring: *Rename Method*

Reasons for doing this

- The old name was not clear
- The task of a method has changed

Seems trivial, but is very important

- Bad naming defeats the whole purpose of encapsulation
- Code is confusing; in the worst case it can be completely misinterpreted

Refactoring: *Replace Magic Number with Symbolic Constant*

Encapsulates a standard piece of programming advice

Applies to strings etc, as well as numbers

Refactoring: *Replace Error Code with Exception*

Code can be written assuming it works

- It can therefore be much easier to read

Error handling code separated out

Refactoring: *Encapsulate Field*

Make a field (instance variable) private

- Add a getter, and a setter if necessary
- Or, in C# and Delphi make the variable a 'property'

Refactoring: *Remove Control Flag*

Remove a local variable that serves as a control flag to break out of loops

Use Break, Continue and Exit instead

Refactoring: *Split Complex Condition*

```
if ((a || b) && (c || d)) {  
    ...  
}
```

Becomes

```
if (a || b) {  
    if (c || d) {  
        ...  
    }  
}
```

The inverse of this can sometimes also be useful

Refactoring: *Name The Condition*

```
if ((a || b) && (c || d)) {  
    ...  
}
```

Becomes

```
meaningfulVariable = a || b;  
anotherVariable = c || d;  
if (meaningfulVariable && anotherVariable) {  
    ...  
}
```

General Considerations about Refactoring 1

Do not apply refactorings without thought for the consequences

- Multi-threaded applications may experience unexpected effects
- Changing interfaces may affect a published API
- Some code is beyond all help

General Considerations about Refactoring 2

Perform good upfront design

- but you can design for ease of refactoring

Don't make optimisation decisions before finding bottlenecks

- Refactor first, optimise after