CSI1102
Introduction to Software Design

Chapter 2:
Objects and Primitive Data

Learning objectives: Objects and Primitive Data

- Introducing objects and their properties
  - Predefined objects: System.out
- Variables and assignment
- Primitive data types
  - Definition of the 8 types
  - Working with arithmetic expressions: operator precedence and data conversion
- Creating Objects in Java (using String examples)
- Class libraries and packages: Random, Math, Keyboard
- Introducing Java applets

Introduction to Objects

- An *object* represents something with which we can interact in a program
- An object **provides a collection of services** that we can tell it to perform for us
- The services are **defined by methods** in a *class* that defines the object
- A **class represents a concept, and an object represents the embodiment of a class**
- A class can be used to create multiple objects

A Predefined Object

- The `System.out` object represents a destination to which we can send output
- A service is performed for us
- In the `Hello` program, we invoked the `println` method of the `System.out` object:

```
System.out.println ("Hello World. My name in Suzy");
```

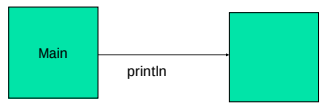
object — method — information provided to the method (parameters)

The print versus the println Methods

```
// Countdown.java (p.65)
// demonstrate difference between print and println

public class Countdown
{
  // prints lines of output representing a rocket countdown
  //
  public static void main (String[] args)
  {
    System.out.print("Three...");
    System.out.print("Two...");
    System.out.print("One...");
    System.out.print("Zero...");

    System.out.println("Liftoff!");
    System.out.println("Houston, we have a problem.");
  }
}
```


The print versus the println Methods

- The `print` **method** is similar to the `println` **method**, except that it does not advance to the next line
- Both **methods** form part of the `System.out` **object**
- **Methods** are invoked by means of **parameters**, which is a "message" sent to the method.



So what are Objects and Classes?

A class (the concept)

Bank Account

An object (the realization)

John's Bank Account
Balance: $5,257

Bill's Bank Account
Balance: $1,245,069

Mary's Bank Account
Balance: $16,833

Multiple objects from the same class

Object orientation definitions

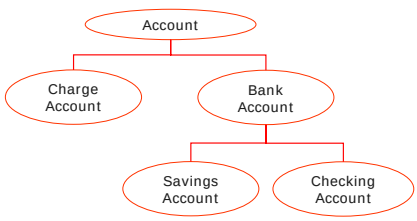
Object orientation is a set of principles that is based on the idea of conceptually autonomous, independent structures called *objects*

Object: Waiter at Restaurant
Actions: Communicate with himself (?!!?), clients and personnel

An Object is an abstract representation of a real world entity with a unique identity, built-in properties. It can communicate with itself as well as other objects.

Object property 1: Inheritance

- One class can be used to derive another via *inheritance*
- Classes can be organized into inheritance hierarchies



Object property 2: Abstraction

- An *abstraction* hides (or suppresses) the right details at the right time
  - → we do not know its internal details
- An object is abstract in that we don't have to think about its internal details in order to use it, i.e. it is a **black box**
- For example, we don't have to know how the `println` method works in order to invoke it

Object property 2: Abstraction

- Why abstraction?
  - A human being can manage only seven (plus or minus 2) pieces of information at one time
    - But if we group information into chunks (such as objects) we can manage many complicated pieces at once
- Classes and objects help us write complex software

Example Predefined Object: Strings (See Facts.java p.68)

```
// Facts.java
// Demonstrate the use of string concatenation and automatic conversion
public class Facts
{
    // print "good to know" facts
    public static void main (String[] args)
    {
        // concatenate two strings into one
        System.out.println("We present the following facts for your " + "use: ");

        // A string containing numeric digits
        System.out.println("Letters in Hawaiian alphabet: 12" );

        // concatenating numeric values to a string
        System.out.println ("Dialing code for Italy:" + 81);
    }
}
```

Example Predefined Object in Java: Character Strings

- Every character string is an object in Java, defined by the `String` class
- Every string literal, delimited by double quotation marks, represents a `String` object
- The *string concatenation operator* (+) is used to append one string to the end of another
- It can also be used to append a number to a string
- **A string literal cannot be broken across two lines in a program**

13

Example Object in Java (cont): String concatenation versus Arithmetic Addition

```
// Addition.java
// Demonstrate the difference between string concatenation and
   arithmetic addition
public class Addition
{
    // Concatenate and adds two numbers and print the result
    public static void main (String[] args)
    {
        System.out.println("24 and 60 concatenated:" + 24 + 60);
        System.out.println("24 and 60 added:" + (24 + 60));
    }
}
```

14

Example Object in Java (cont): String concatenation versus Arithmetic Addition

- The plus operator (+) is also used for arithmetic addition
- The function that the + operator performs depends on the type of the information on which it operates
- If **both operands are strings**, or if **one is a string** and **one is a number**, it performs **string concatenation**
- If both operands are **numeric**, it **adds them**
- The **+ operator is evaluated left to right**
- Parentheses can be used to force the operation order

15

More about strings: Escape Sequences

- What if we wanted to print a double quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\), which indicates that the character(s) that follow should be treated in a special way

```
System.out.println ("I said \"Hello\" to you.");
```

16

More about strings: Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

- See Roses.java (page 71)

17

Using Variables in Java

- A *variable* is a name for a location in memory
- A variable must be *declared* by specifying the variable's name and the type of information that it will hold

data type **variable name**

```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

18

Variables and Assignments

```
// Geometry.java
// Illustrate the use of variable assignments
public class Geometry
{
        // Prints the number of sides on a geometric shape
        public static void main (String[] args)
        {
            int sides = 7;
            System.out.println(" A hegtagon has " + sides + " sides".);
            sides = 12;
            System.out.println(" A dodegan has " + sides + " sides".);
        }
}
```


Variables and Assignments

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used
- An *assignment statement* changes the value of a variable `total = 55;`
- You can assign only a value to a variable that is consistent with the variable's declared type

Constants

- A constant is an identifier that is similar to a variable except that it **holds one and only one value** while the program is active
- The compiler will issue an error if you try to change the value of a constant during execution
- In Java, we use the `final` modifier to declare a constant `final int MIN_HEIGHT = 69;`
- Constants:
  - give names to otherwise unclear literal values
  - facilitate updates of values used throughout a program
  - prevent inadvertent attempts to change a value

Primitive Data Types

- There are exactly **eight** primitive data types in Java
- Four of them represent integers:
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers:
  - `float`, `double`
- One of them represents characters:
  - `char`
- And one of them represents boolean values:
  - `boolean`

Primitive data types 1-6: Numeric

- The difference between the various numeric primitive types is their size, and therefore the values they can store:

Type	Storage	Min Value	Max Value
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	< -9 x 10^{18}	> 9 x 10^{18}
`float`	32 bits	+/- 3.4 x 10^{38} with 7 significant digits	
`double`	64 bits	+/- 1.7 x 10^{308} with 15 significant digits	

Binary Math (revision)

- Once information is digitized, it is represented and stored in memory using the *binary number system*
- A single binary digit (0 or 1) is called a *bit*
- Devices that store and move information are cheaper and more reliable if they have to represent only two states
- A single bit can represent two possible states, like a light bulb that is either on (1) or off (0)
- Permutations of bits are used to store values

Bit Permutations

1 bit	2 bits	3 bits	4 bits
0	00	000	0000 1000
1	01	001	0001 1001
	10	010	0010 1010
	11	011	0011 1011
		100	0100 1100
		101	0101 1101
		110	0110 1110
		111	0111 1111

Each additional bit doubles the number of possible permutations

Bit Permutations (cont)

- Each permutation can represent a particular item
- There are 2^N permutations of N bits
- Therefore, N bits are needed to represent 2^N unique items. One bit could be used for sign (negative numbers).

How many items can be represented by

1 bit?
2 bits?
3 bits?
4 bits?

Primitive data type 7: Characters

- A `char` variable stores a single character from the *Unicode character set*
- A *character set* is an ordered list of characters, and each character corresponds to a unique number
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters
- It is an international character set, containing symbols and characters from many world languages
- Character literals are delimited by single quotes:

```
'a'   'X'   '7'   '$'   ','   '\n'
```


Primitive data type 7: Characters (cont)

- The *ASCII character set* is older and smaller (uses 7 bits) than Unicode, but is still quite popular
- The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	**A, B, C, …**
lowercase letters	**a, b, c, …**
punctuation	**period, semi-colon, …**
digits	**0, 1, 2, …**
special symbols	**&, |, \, …**
control characters	**carriage return, tab, ...**

Primitive data type 8: Boolean

- A `boolean` value represents a true or false condition
- A `boolean` also can be used to represent any two states, such as a light bulb being on or off
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```


Using primitive data: Arithmetic Expressions

- An *expression* is a combination of one or more operands and their operators
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	**+**
Subtraction	**-**
Multiplication	*****
Division	**/**
Remainder	**%**

- If either or both operands associated with an arithmetic operator are floating point, the result is a floating point

Using primitive data: Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3      equals?

8 / 12      equals?
```

- The remainder operator (%) returns the remainder after dividing the second operand into the first

```
14 % 3      equals?

8 % 12      equals?
```

31

Primitive data Operator Precedence

- Operators can be combined into complex expressions

```
result  =  total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated
  - Multiplication (*), division (/), and remainder (%) are evaluated prior to
    - addition (+), subtraction (-), and string concatenation (+)
  - Arithmetic operators with the same precedence are evaluated from left to right
  - Parentheses can be used to force the evaluation order

32

Operator Precedence

- What is the order of evaluation in the following expressions?

```
a + b + c + d + e        a + b * c - d / e

a / (b + c) - d % e

a / (b * (c + (d - e)))
```

33

Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
```

Then the result is stored in the variable on the left hand side

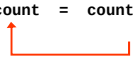
34

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of count

```
count  =  count + 1;
```



Then the result is stored back into count (overwriting the original value)

35

Data Conversions

- For example, we may want to treat an **integer** as a **floating point** value during a computation
- Conversions must be handled carefully to avoid losing information
- In Java, data conversions can occur in three ways:
  - assignment conversion
  - arithmetic promotion
  - casting

36

Data conversions: Widening versus Narrowing

byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

Data Conversions

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
  - **Only widening conversions** can happen via assignment

```
Money = dollars;     // int to float
```

- *Arithmetic promotion* happens automatically when operators in expressions convert their operands

```
result = sum/count;  // float = float/int
```


Data Conversions

- *Casting* is the most powerful, and dangerous, technique for conversion
  - Both widening and narrowing conversions can be accomplished by explicitly casting a value
  - To cast, the type is put in parentheses in front of the value being converted
- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:

```
result = (float) total / count;
```


Creating Objects

- A variable holds either a **primitive type** or a ***reference* to an object**
- A class name can be used as a type to declare an ***object reference variable***

```
String title;
```

- **No object is created** with this declaration
- An object reference variable holds the **address** of an object
- The **object itself must be created separately**

Creating Objects

- Generally, we use the `new` operator to create an object

```
title = new String ("Java Software Solutions");
```

This calls the String *constructor*, which is a special method that sets up the object

- Creating an object is called ***instantiation***
- An object is an *instance* of a particular class

Creating Objects

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

```
title = "Java Software Solutions";
```

- This is special syntax that works **only for strings**
- Once an object has been instantiated, we can use the *dot operator* to invoke its **methods**

```
title.length()

System.out.println()
```

String Methods

- The `String` class has several methods that are useful for manipulating strings, including
  - length
  - toLowercase
  - Substring
- See p.89 and Appendix M of the text book for others
- Many of the methods *return a value*, such as an integer or a new `String` object

An example use of String methods

```
// StringMutation.Java
public class StringMutation
{
    public static void main (String[] args)
    {
        String phrase = new String ("Change in unavoidable");
        String mutation1, mutation2, mutation3, mutation4;

        mutation1 = phrase.concat(", except for some machines.");
        mutation2 = mutation1.toUpperCase();
        mutation3 = mutation2.replace('O', 'Z');
        mutation4 = mutation3.substring(3, 12);

        System.out.println("Final string: " + mutation4);
        System.out.println("Final string length: " + mutation4.length());
    }
}
```


Java Class Libraries

- A *class library* is a collection of classes that we can use when developing programs
- The *Java standard class library* is part of any Java development environment
- Its classes are not part of the Java language *per se*, but we rely on them heavily
- The `System` class and the `String` class are part of the **Java standard class library**
- **Other class libraries can be obtained through third party vendors, or you can create them yourself**

Java Class Libraries

- Classes of the Java Standard Class Library are grouped into **Packages**, i.e. related classes
- A cluster of related classes are called **Java APIs**, or Application Programming Interfaces,
  - e.g. the Java Database API, Java Swing API
- A given API may consist of classes from various packages
- See Appendix M of the text book

Packages

- Some of the packages in the standard class library are:

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities and components
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

- See Appendix M

The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Random
```

- Or you can *import* the class, and then **use just the class name**

```
import java.util.Random;
```

- To import all classes in a particular package, you can use the * wildcard character

```
import java.util.*;
```

The import Declaration

- All classes of the `java.lang` package are imported automatically into all programs
- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs
  - For example, the `Random` class is part of the `java.util` package
  - It provides methods that generate pseudorandom numbers

49

An example: Importing the Random Class

```
// Randomnumbers.java
//
import java.util.Random;

public class RandomNumbers
{
    public static void main (String[] args)
    {
        Random generator = new Random();
                        //generator is an object of class Random
        int num1;

        num1 = generator.nextInt();
        System.out.println("A random integer: " + num1);

        num1 = generator.nextInt(15);
        System.out.println("From 0 to 14: " + num1);
    }
}
```

50

Class (Static) Methods

- Some methods can be invoked through the **class name**, instead of through **an object of the class**
- These methods are called *class methods* or *static methods*
- The `Math` class (see figure 2.13 on p.99) contains many static methods, providing various **mathematical functions**, such as absolute value, trigonometry functions, square root, etc.

```
temp = Math.cos(90) + Math.sqrt(delta);
```

51

The Keyboard Class

- The `Keyboard` class is NOT part of the Java standard class library; It is provided by the authors of the textbook to make reading input from the keyboard easy
- The `System.in` object is part of the Java Standard library; as discussed in **Chapter 8**
- Details of the `Keyboard` class are explored in **Chapter 5**
- The `Keyboard` class is part of a package called **`cs1`**
- It contains several **static methods** for reading particular types of data
- Also See Quadratic.java (page 102)

52

The Keyboard Class

```
// Echo.java
// Demonstrate ReadString and ReadInt methods of Keyboard class
import cs1.Keyboard;

public class Echo
{
    // read from the user and print it out
    public static void main (String[] args)
    {
        String message;
        int a;

        System.out.println("Enter any text: ");
        message = Keyboard.readString();

        System.out.println("Enter any integer: ");
        a = Keyboard.readInt();

        System.out.println("You entered: \"" + message + "\"" + " and " + a);
    }
}
```

53

Formatting Output

- The `DecimalFormat` class can be used to format a floating point value in generic ways
- For example, you can specify that the number should be printed to three decimal places
- The constructor of the `DecimalFormat` class takes a string that represents a pattern for the formatted number
- See CircleStats.java (page 107)
- Also see the `NumberFormat` class on p.103

54

Introduction to Applets

- A Java **application** is a **stand-alone program with a `main` method** (like the ones we've seen so far)
  - A Java ***applet*** is a program that is intended to transported over the Web and executed using a web browser
  - An applet also can be executed using the appletviewer tool of the Java Software Development Kit
  - **An applet doesn't have a `main` method**
  - Instead, there are several special methods that serve specific purposes

An example Applet: Einstein.java

```
// Demonstrate a basic Applet
import java.applet.Applet;
import java.awt.*;

public class Einstein extends Applet
{
   // Draw a quotation and some shapes
   public void paint (Graphics page)
   {
      page.drawRect(50, 50, 40, 40); //square
      page.drawOval(75,65, 20, 20); //circle
      page.drawString("Imagination is more important than knowledge.", 110, 70);
      page.drawString("- Albert Einstein", 130, 100);
   }
}
```


Discussion of the Einstein.java Applet

- The class that defines an applet *extends* the `Applet` class
  - This makes use of *inheritance*, which is explored in more detail in Chapter 7
- The `paint` method, for instance, is executed automatically and is used to draw the applet's contents
- The `paint` method accepts a **parameter** that is an object of the **`Graphics` class**
- A `Graphics` object defines a *graphics context* on which we can draw shapes and text
- The `Graphics` class has several methods for drawing shapes

How are Applets Executed?

- An applet is embedded into an HTML file using a tag that references the bytecode file of the applet class
- The bytecode version of the program is transported across the web and executed by a Java interpreter that is part of the browser
- See Appendix J for more information

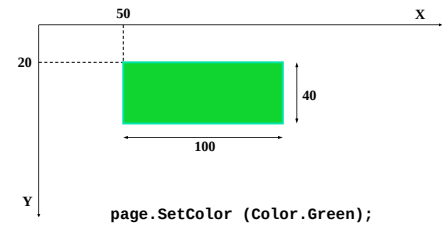
The HTML applet Tag

```
<html>

   <head>
      <title>The Einstein Applet</title>
   </head>

   <body>
      <applet code="Einstein.class" width=350 height=175>
      </applet>
   </body>

</html>
```


Drawing Shapes

- The `Graphics` class draw shapes in more detail
- A shape can be filled or unfilled, depending on which method is invoked
- The method parameters specify coordinates and sizes
- The Java coordinate system has the origin in the top left corner
- Shapes with curves, like an oval, are usually drawn by specifying the shape's *bounding rectangle*
- See Figure 2.18 (p.112) for some methods

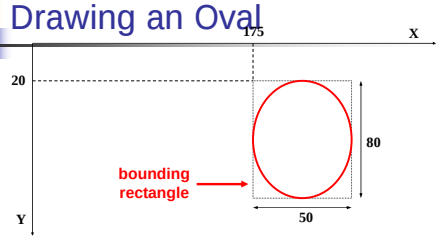
For Example: Drawing a Rectangle




```
page.SetColor (Color.Green);
page.fillRect (50, 20, 100, 40);
```

See Snowman.java on page 115

Drawing an Oval




```
page.drawOval (175, 20, 50, 80);
```


Summary: Chapter 2

- predefined objects
- primitive data
- the declaration and use of variables
- expressions and operator precedence
- creating and using objects
- class libraries
- Java applets
- drawing shapes