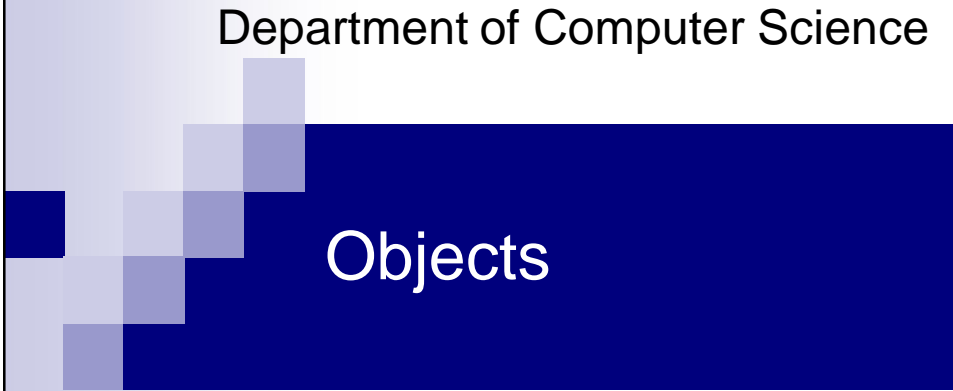



Department of Computer Science



Objects

Dr. Chia-Ling Tsai



Outline

- Idea of an object
- Constructing objects
- Library objects
- Objects for input and output
- Randomizers
- Object references

Idea Of An Object

- An *object* is something you can use without specific knowledge of its internal workings
- You manipulate an object by *sending it a message*
 - Also known as *calling* one of its *methods*
 - The *static methods* we have already used and written are different – there is no object involved
- A *class* is the implementation of an object.
- An object is *referenced by a variable* (instance) of the class

Example—String Objects

```

// A program to demonstrate the object concept
public class StringObject {
    public static void main(String[] args) {
        String s1 = "Hello World!";
        int len = s1.length();
        System.out.println(len);
        String s2 = s1.toUpperCase();
        System.out.println( s2 );
    }
}

```

Class name

Object instances

methods

What We'll Do

- Just as for programs and methods, classes have *users* and *implementers*
 - For example, we can call a static method (use) but must also define it (implement)
- Similarly, a class must be implemented to define the behavior of the object.
 - You can think the class implementation as the blueprint of the object
- *We will not implement classes in this course*

Familiar Objects

- `System.out` is an object to which you can send a message to print things (such as strings)
 - Both `print` and `println` are messages asking for something to be printed (the second one also goes to the next line)
 - You tell Java the information you want printed in the parentheses
`System.out.println(greeting);`
- You can look up information in the [Java library documentation](http://java.sun.com/j2se/1.4.2/docs/api/) : <http://java.sun.com/j2se/1.4.2/docs/api/>
- Often, there is more information than you need – or want – but you can usually find what you're after by checking the names and descriptions
 - The textbook, in Appendix B, has less complete but also easier to read documentation for this and other classes

Familiar Objects

- An individual string is an object of class `String`
 - Defined in `java.lang`
 - One of the messages a string accepts is `length`
 - You send the message using a period between the object and the message

```
String greeting = "how are you?";
int greetingSize = greeting.length();
```

0	1	2	3	4	5	6	7	8	9	10	11
h	o	w		a	r	e		y	o	u	?

Case Study: The encoding problem again

```
import java.util.*;
/*
Algorithm:
The problem consists of the following steps:
1. Read in the input" row, column, and the char string
2. Convert the character string to binary string
3. Convert the 1D binary vector to a 2D matrix in a spiral manner
4. Convert the 2D matrix to 1D vector again by concatenating in row-major order
*/
public class encoding {
    public static void main(String[] args) {
        // Step1: Read the input from the keyboard
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter the number of rows: ");
        int row = keyboard.nextInt();
        System.out.print("Enter the number of columns: ");
        int col = keyboard.nextInt();
        String garbage = keyboard.nextLine();
        System.out.print("Enter the string to be encoded: ");
        String char_input = keyboard.nextLine();

        // Step2: Convert the character string to binary string
        String binary_input = char_to_binary(char_input);

        // Step3: Convert the 1D binary vector to a 2D matrix in a spiral manner
        // Step4: Convert the 2D matrix to 1D vector again by concatenating in row-major order
    }

    static String char_to_binary(String input)
    {
        -----
    }
}
```

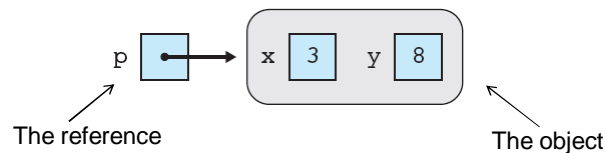
Constructing Objects

- `System.out` is an object that is “already there,” provided by the Java library, and strings are special: we can make new ones using quotation marks (and also `+` between strings we already have made)
- Objects are usually made (*constructed*) using the `new` keyword
 - `String` is an exception
 - *Example:* We do this when we construct a scanner for input


```
Scanner keyboard = new Scanner(System.in);
```
- To know how to use these objects once “constructed,” we must consult *documentation*

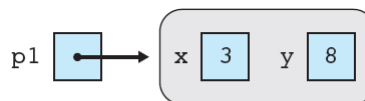
A More Typical Example

- The Java library has a class called `Point`
- You “construct” a point by using `new` and providing the coordinates of the point
 - Construction involves creation and initialization
- *Example:* `Point p = new Point(3, 8);`
- Defined in `java.awt`



Assignment of Objects

- `Point p1 = new Point(3,8);`
- `Point p2 = new Point();` ← Default constructor
- `Point p3 = p2;`



Another Example

```
import java.awt.Point;
// A sample program showing how to use Point objects
public class UsingPoints
{
    // The program
    public static void main(String[] args)
    {
        Point p1 = new Point(10, 20);
        Point p2 = new Point(20, 10);

        printPointInfo("At the beginning, p1 is ", p1);
        printPointInfo("At the beginning, p2 is ", p2);

        p1.translate(50, 50);
        p2.translate(30, 30);

        printPointInfo("After translating it, p1 is ", p1);
        printPointInfo("After translating it, p2 is ", p2);
    }

    // Method that prints information about a point
    static void printPointInfo(String message, Point p)
    {
        System.out.println(message + "(" + p.x + ", " + p.y + ")");
    }
}
```

Class Point

`java.awt`

- `java.lang.Object`
 - `java.awt.geom.Point2D`
 - `java.awt.Point`

All Implemented Interfaces:

- `Serializable`
- `Cloneable`

Annotations:

- `import java.awt.Point;` → `java.awt`
- `Point p1 = new Point(10, 20);` and `Point p2 = new Point(20, 10);` → Construct objects
- `p1.translate(50, 50);` and `p2.translate(30, 30);` → Send messages to objects (Call object methods)
- `printPointInfo("After translating it, p1 is ", p1);` and `printPointInfo("After translating it, p2 is ", p2);` → Objects as arguments
- `printPointInfo(String message, Point p);` → Object parameter
- `p.x` and `p.y` → Use object attributes

Library Objects

- We have already used objects from the Java library
 - Strings
 - Scanners
 - Points
 - `System.in` and `System.out`
- Other objects are available, for graphics, internet programming, and a lot more
 - Make good use of the documentation
- We'll look at a few more here

Objects for Input And Output

- `System.in` and `System.out` are objects already constructed and made available from the `System` class, for convenience
- Because `System.in` isn't so convenient, we have used the `Scanner` class for easier input
- `System.out` works better
 - It has `print`, `println`, and `printf` methods
 - These methods actually disguise the use of other classes – `PrintStream` for the first two methods and `Formatter` for the third
- There are many other I/O classes in the library too, for graphics (`JOptionPane`), the internet (`HTMLDocument`), etc

Randomizers – Basics

- Another very useful class from the Java library is `Random`
- Choosing numbers “at random” is important when we try to *simulate* events
- Computers do what they are told, so they can't really select “at random”
- Instead, computers use *pseudorandom numbers*, which are sequences of numbers that
 - *Appear* to be random
 - are *distributed* uniformly enough to appear random

Randomizers – Specifications

- The Java library class called `Random` can produce pseudorandom numbers
- Once we have a randomizer, we can use it to
 - Get a “random” decimal between 0.0 and 1.0, using `nextDouble` method
 - Get a “random” integer between 0 and a number we choose using the `nextInt` method
 - `nextInt(10)` gives one of the numbers 0,1,2,3,4,5,6,7,8,9
- Notice the similarity in method names to those of the `Scanner` class

Using Randomizers

- Although the methods in class `Random` are restrictive, we can use them to get numbers in any range
- The method `nextDouble` returns a number in the range 0.0 to 1.0. To get a number in the range `a` to `b`, use
 $a + (b-a)*nextDouble()$
- The method `nextInt` accepts an argument, `n`, and returns a number in the range 0 to `n-1`, inclusive. To get a number in the range `a` to `b`, inclusive, use
 $a + nextInt(b-a+1)$

Example

Output
will be
different
every
time

```
import java.util.Random;
// Sample program illustrating uses of class Random
public class UseRandomizer
{
    public static void main(String[] args)
    {
        Random randomizer = new Random();

        System.out.println("Random numbers between 0 and 1");
        for (int i = 1; i <= 20; i++)
        {
            double x = randomizer.nextDouble();
            System.out.println(x);
        }
        System.out.println("Random numbers between 0 and 99, inclusive");
        for (int i = 1; i <= 20; i++)
        {
            int n = randomizer.nextInt(100);
            System.out.println(n);
        }
        System.out.println("Random numbers between 1 and 10");
        for (int i = 1; i <= 20; i++)
        {
            double x = randomizer.nextDouble(); // ← A number from 0 to 1
            double y = 1 + 9*x; // ← From 1 to 10
            System.out.println(y);
        }
        System.out.println("Random numbers between 500 and 1000, inclusive");
        for (int i = 1; i <= 20; i++)
        {
            int n = randomizer.nextInt(501); // ← From 0 to 500
            int m = 500 + n; // ← From 500 to 1000
            System.out.println(m);
        }
    }
}
```

Object References

- Variables used for objects *refer* to the objects
- Each object has its own *identity*
 - You can think it as the street address of a house
- Each time you use the **new** keyword, a new object, with its own identity, is constructed
- *If you don't use the new keyword, you don't get a new object – just a new reference*
- Primitive types do *not* use the **new** keyword
 - You just declare the variable and initialize it with a value
 - Each variable *labels* a different value

Example: Primitive Data

```
int x = 100;
int y = x;    // a DIFFERENT number than x
int z = 100; // also different

// Display information
System.out.println("NUMBERS AT START:");
System.out.println("x is " + x);
System.out.println("y is " + y);
System.out.println("z is " + z);

// Change one of the numbers
x = x + 50; // does NOT affect y

// Display changed information
System.out.println("NUMBERS AFTER CHANGE TO x:");
System.out.println("x is " + x);
System.out.println("y is " + y);
System.out.println("z is " + z);
```

```
NUMBERS AT START:
x is 100
y is 100
z is 100
NUMBERS AFTER CHANGE TO x:
x is 150
y is 100
z is 100
```

Example: Objects

```

Point P = new Point(100, 200);
Point Q = P; // refers to SAME point as P
Point R = new Point(100, 200); // refers to a DIFFERENT point
                                // (with same coordinates)

// Display information about points
System.out.println("POINTS AT START:");
System.out.println("The x-coordinate of P is " + P.x);
System.out.println("The x-coordinate of Q is " + Q.x);
System.out.println("The x-coordinate of R is " + R.x);

// Change the coordinates in one point
P.translate(50, 40); // affects Q too!

// Display changed information
System.out.println("POINTS AFTER CHANGE TO P:");
System.out.println("The x-coordinate of P is " + P.x);
System.out.println("The x-coordinate of Q is " + Q.x);
System.out.println("The x-coordinate of R is " + R.x);

```

```

POINTS AT START:
The x-coordinate of P is 100
The x-coordinate of Q is 100
The x-coordinate of R is 100
POINTS AFTER CHANGE TO P:
The x-coordinate of P is 150
The x-coordinate of Q is 150
The x-coordinate of R is 100

```

Pass by Reference

- We have done pass-by-value in the previous series
 - The variable which is passed to the method keeps its original value
- If you pass a reference to a method, it is the reference that is passed, not the value!

```
import java.awt.Point;
// Simple program to demonstrate parameter passing methods
public class testPoint {
    public static void main(String[] args) {
        Point p=new Point(5,10);
        test(p);
        System.out.println("p.x = "+p.x +", p.y = "+p.y );

        Point q=new Point(20, 40);
        test2(q.x,q.y);
        System.out.println("q.x = "+q.x +", q.y = "+q.y );
    }
    // Pass by reference
    static void test(Point p) {
        p.translate(50,100);
    }
    // Pass by value
    static void test2(int x, int y) {
        x += 50;
        y += 100;
    }
}
```

Summary

- The idea of an object and its construction
- Library objects
- Object references