Introduction to Objects
and
Object-Oriented Programming
Agendas

– All about Object!
– Encapsulation.
– Message Passing.
– Input/Output.
– Syntax Errors Vs. Logic Errors.
– Selection and Iteration.
An object is a "thing," a “gizmo," a "gadget," … an object.

For example, a car, a soda machine, a dog, a person, a house, a bank account, a pair of dice, a deck of cards, a point in the plane, a TV, a VCR, an ATM machine, an elevator, a square, a circle, a flea, an elephant, a camera, a movie star, a computer mouse, a live mouse, a phone, an airplane, a song, … just about anything is an object.

In computing, a window is an object, so is a mouse, a menu, a textbox, and a button.

Objects come in all shapes, sizes, and colors. An object may be physical, like a radio, or intangible, like a song.

For our purposes, however, objects are entities that have

1. attributes, (characteristics or properties), and
2. methods, (actions or behaviors of an object ).
Elevator Object

- Notice that the three elevator objects have different attribute values.
  
  The attribute values determine the state of an object.

- Thus the state of elevator 1 is that the floor is 3 and the door is open.

- The state of elevator 3 is that the current floor is 2 and the door is closed.

- Each elevator object has a unique state.

- On the other hand, all elevator objects have the same behavior.

- However, all three objects can do the same things (open the door, close the door etc.).

- All three have the same methods.
Rectangle Object

• A rectangle is an object
• Some possible properties or attributes of a rectangle are:
  1. length and
  2. width
• Some possible methods are:
  1. get the length
  2. get the width
  3. change the length
  4. change the width
  5. get the area
  6. get the perimeter

Here are two rectangle objects:

• The state of the first rectangle object is \{length = 7, width = 5\} ;
• The state of the second rectangle is \{length = 2, width = 8\}.
Computer Widow Object!

- **A computer window** is an object
  - Some of the (many) attributes include:
    1. length
    2. width
    3. background color
    4. font style
    5. font color
    6. state – maximized, minimized or downscaled
  - Some of the (many) methods include:
    1. resize the window (change length and width)
    2. maximize
    3. minimize
    4. change background color
    5. change font etc.
Defining Object

• In the context of a computer program, you might think of an object as a representation, model or abstraction of some entity consisting of
  1. data (attributes) and
  2. functions (methods) which use or manipulate the data

• An object’s data determines its state. For example, the data for the first elevator (above) specifies that the “floor” attribute has value 3 and the door is open. The current state of the first rectangle indicates that the length of the rectangle is 7 and the width is 5.

• The methods/functions specify what an object does, i.e., the behavior of an object.
How do you define Object?

- The attributes and methods of an object depend on our specific use and view of an object.
- For example, in one application, a rectangle object might be a simple geometrical figure with just two attributes, length and width.
- Another, perhaps graphical, view of a rectangle might include color and location (x and y coordinates) among the attributes.
- Similarly, an elevator has many potential attributes (carpet color, number of passengers, maximum weight, date of last inspection) but only a few attributes are of interest in any application.
Object has Object!

• Some attributes themselves might be other objects.
• For example, a light object may have:
  – attributes:
    • number of watts
    • current state (on or off)
  – methods:
    • turn light on
    • turn light off
• Now a light object may be part of (an attribute of) an elevator object.
Your Turn!

A circle is an object.

<table>
<thead>
<tr>
<th>attribute (data) of a circle:</th>
<th>The methods (functions) might be</th>
</tr>
</thead>
<tbody>
<tr>
<td>–radius, a real number.</td>
<td>a. give (return) its area</td>
</tr>
<tr>
<td></td>
<td>b. give (return) its circumference.</td>
</tr>
</tbody>
</table>

In each of the examples, the attributes and methods have been chosen arbitrarily. Indeed, choosing the “right” attributes and methods for an object is a skill and an art that comes with practice and patience.
**A Bank account is an object.**

<table>
<thead>
<tr>
<th>Data or Attributes of a bank account might be</th>
<th>The methods/functions/operations might be</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. an ID number</td>
<td>a. give (return) the balance</td>
</tr>
<tr>
<td>b. customer's name,</td>
<td>b. give personal information about the account owner</td>
</tr>
<tr>
<td>c. address, and</td>
<td>c. make a deposit</td>
</tr>
<tr>
<td>d. balance</td>
<td>d. make a withdrawal</td>
</tr>
</tbody>
</table>
What is Classes?

• A class is a template, blueprint, or description of a group of objects.
• Every object is described by some class. For example,
• An elevator class specifies the characteristics and behaviors of all elevator objects. The elevator class is a general description of an elevator. An elevator class is not an elevator.
• A rectangle class describes the attributes and methods of all rectangle objects. A rectangle class may specify that every rectangle object has both a length and a width. However, a rectangle class is not a rectangle.
An analogy involves comparing two things to highlight similarities. Here’s a look at the analogy between an architect’s blueprint and a class in programming:

1. **An architect’s blueprint** is analogous to a **class**. A blueprint is not a house but a description or specification of a potential house.

2. **When a builder constructs** two real houses from a blueprint, well, **now** we have two “house objects.”

3. **The skill of the programmer** in defining classes is akin to the **skill of the architect**, and the **labor of the compiler** in building objects is like the **work of the construction company**.
Object Analogy
An object is an instance of a class!

• Just as a builder creates houses from a blueprint,
• From one blueprint, a builder can build many houses.
• Every house builds from the blueprint

• A program creates objects from a class.
• From one class, a program can create many objects
• Every object is “manufactured” according to some class specification.
• Every object belongs to some class.
Class Example

- Let us look at an example of a Rectangle class in Java.
- The class is a description (in Java) of the attributes and behaviors of a rectangle object.
- For now, don’t be concerned with the syntax or any of the Java particulars.
Rectangle Class

public class Rectangle
{
  //Every Rectangle object has both length and width attributes (int)
  private int length;
  private int width;
  //default values for a rectangle object are length = 1 and width = 1
  public Rectangle() // default constructor
  {
    length = 1;
    width = 1;
  }
  //can create a Rectangle object with any dimensions
  public Rectangle(int x,int y) //constructor
  {
    length = x;
    width = y;
  }
  //can change the dimensions of any rectangle object
  public void changeDimensions(int x,int y) // mutator
  {
    length = x;
    width = y;
  }
  //gives the area of a rectangle object // accessor
  public int getArea()
  {
    return length*width;
  }
  //gives the perimeter of a rectangle object
  public int getPerimeter()
  {
    return 2*(length+width);
  }
}
Analogy of the Code

• The preceding code (a Rectangle class) is a \textit{template} for a rectangle. According to the specifications, any potential rectangle object has both a \textbf{length} and a \textbf{width} of type int. Moreover any rectangle object can
  • change its dimensions,
  • give its area, and
  • give its perimeter.

• OK, we know what a rectangle object has and what it can do. So, we can now manufacture, create, instantiate as many rectangles as we like. We have the blueprint, so let’s start production:

  \begin{verbatim}
  // makes a 5 X 7 rectangle named r1
  Rectangle r1 = new Rectangle(5,7);
  // makes a 7 X 5 rectangle named r2
  Rectangle r2 = new Rectangle(7,5);
  // makes a default 1 X 1 rectangle named r3
  Rectangle r3 = new Rectangle();
  \end{verbatim}
Object

• With three magic statements, we have created three rectangle objects --- built according to the specifications of our class/blueprint.

• Each rectangle object has a length property and a width property with appropriate values:
OOP Concept: Encapsulation

• Webster defines encapsulation as being “enclosed by a capsule.”
• Real world examples of encapsulation surround us:
• A computer is an example of real world encapsulation. The chips, boards, and wires of a computer are never exposed to a user. Like the TV viewer, a computer user operates a computer via an interface -- a keyboard, screen, and pointing device.
Example: Encapsulation

• A cabinet hides (encapsulates) the “guts” of a television, concealing from TV viewers the internal apparatus of the TV.

• Moreover, the television manufacturer provides users with an interface --the buttons on the TV or perhaps a remote control unit.

• To operate the TV and watch *The Simpsons* or *Masterpiece Theatre*, viewers utilize this interface. The inner circuitry of a TV is of no concern to most TV viewers.

• Cameras, vending machines, slot machines, DVD players, lamps, cars, video games, clocks, and even hourglasses are all physical examples of encapsulation.
Encapsulation: Explained

• Each of the devices enumerated above supplies the user with an interface — switches, buttons, remote controls, whatever -- for operation.
• Technical details are tucked away and hidden from users.
• Each encapsulated item functions perfectly well as a “black box.”
• Certainly, Joe User need not understand how his Radio Shack gadget is constructed in order to operate it correctly.
• A user-friendly interface and perhaps an instruction manual will suffice.
Encapsulation: Explained

• Encapsulation has a similar (though somewhat expanded) meaning when applied to software development and object oriented programming:

  The ability to provide users with a well-defined interface to a set of functions in a way which hides their internal workings. In object-oriented programming, the technique of keeping together data structures and the methods (procedures) which act on them.

  – The Online Dictionary of Computing

• Java provides encapsulation, as defined above, via classes and objects.

• As we have already seen, classes bundle data and methods into a single unit. Classes encapsulate.
public class Rectangle
{
    /*Every Rectangle object has both length and width attributes (int)*/
    private int length;
    private int width;
    /*default values for a rectangle object are length = 1 and width = 1*/
    public Rectangle() // default constructor
    {
        length = 1;
        width = 1;
    }
    /*can create a Rectangle object with any dimensions*/
    public Rectangle(int x,int y) //constructor
    {
        length = x;
        width = y;
    }
    /*can change the dimensions of any rectangle object*/
    public void changeDimensions(int x,int y) // mutator
    {
        length = x;
        width = y;
    }
    /*gives the area of a rectangle object*/
    /* accessor method */
    public int getArea()
    {
        return length*width;
    }
    /*gives the perimeter of a rectangle object*/
    public int getPerimeter()
    {
        return 2*(length+width);
    }
}
public class TestRectangle{
    public static void main(String args[]){
        // makes a 5 X 7 rectangle named r1
        Rectangle r1 = new Rectangle(5,7);
        // makes a 7 X 5 rectangle named r2
        Rectangle r2 = new Rectangle(7,5);
        // makes a default 1 X 1 rectangle named r3
        Rectangle r3 = new Rectangle();
        System.out.println(r1.getPerimeter());
        double area = r2.getArea();
        r3.changeDimensions(7,3);
    }
}
An object

data field 1
...
data field n
method 1
...
method n

State

Behavior

A Rectangle object

Data Field
length = 1
width = 1

Method
getArea
Encapsulation

• The Rectangle class provides a simple example of encapsulation – data and methods, attributes and functionality, are combined into a single unit, a single class.
• Furthermore, all data are accessed not directly but through the class methods, i.e. via an interface TestRectangle.
• The user of Rectangle – the client -- need not know how the class is implemented – only how to use the class.
• Variable names are of no concern to the client. If the client wishes to know the perimeter of a Rectangle object, the interface provides an accessor method.
Encapsulation

• If the client wants to dimension the size of a Rectangle object, the client simply uses the mutator method available through the interface.

• That the length of the sides of a square is held in a variable called *dimension* is irrelevant to the client.

• Further, if the implementation of the class is modified, programs utilizing the Rectangle class will not be affected provided the interface remains unchanged.

• Like a TV or a camera, the inner workings of the Rectangle class are encapsulated and hidden from the client.

• Public methods provide the interface just as the remote control unit provides the interface for a TV viewer.
Encapsulation: Information Hiding

• Another term that is often associated with encapsulation is *information hiding*.
• Many authors regard encapsulation and information hiding as synonyms.
• However OO purists might define *encapsulation* as the *language feature* allowing the bundling of data and methods into one unit
• and *information hiding* as the *design principle* that restricts clients from the inner workings of a class.
• With this distinction, programs can have encapsulation without information hiding.
• Regardless of how you define your terms, classes should be designed with a well-defined interface in which implementation decisions and details are hidden from the client.
Messages

• In a Java program, objects interact with other objects by sending messages.
• Messages are similar to function calls in procedural style programming.
• The following three statements send messages to r1, r2 and r3, respectively:
  
  System.out.println(r1.getPerimeter());
  area = r2.getArea();
  r3.changeDimensions(7,3);

• The purpose of the messages should be pretty obvious:
  – “r1, get your perimeter!”
  – “r2, get your area!”
  – “r3, change your dimensions!”
Example of Message Passing

- From our perspective a Dog object is a very simple creature:
- Our simple Dog class which has only a single attribute:
  - bark.
- A Dog object can do only two things:
  - set its bark and
  - speak, i.e., bark

```java
public class Dog {
    /*A Dog object has but a single attribute—its bark*/
    private String bark;
    /*set the sound: “woof-woof,” “bow-wow” etc.*/
    public void setBark(String s) {
        bark = s;
    }
    /*Every dog can bark*/
    public void speak() {
        System.out.print(bark);
    }
}
```
Creation of “Fido” and “Brutus”

• We now create instantiates a few Dog objects:
  /* create a Dog named fido*/
  Dog fido = new Dog();
  /*create a Dog named brutus*/
  Dog brutus = new Dog();
• send a few messages to the critters:
  /*a message to fido*/
  fido.setBark(“Bow-Wow”);
  /* a message to brutus*/
  brutus.setBark(“Woof-Woof”);
  /* fido, speak, boy!!*/
  fido.speak();
  /* you too, brutus, speak*/
  brutus.speak();
Input and Output

• Input and output classes model different i/o services
• Setting up i/o involves a composition of constructors
• Example (input):

```java
BufferedReader br = new BufferedReader(
    new InputStreamReader(
        new FileInputStream(new File("USD ata"))));

// file input
```
BufferedReader

• Provides a readLine method
• Returns reference to String modeling line read in OR returns null if end of file is reached
• Setting up i/o involves a composition of constructors
• Example (input):

```java
String s = br.readLine();
while (s!=null) {
    System.out.println(s);
    s = br.readLine();
}
```
/*saved as Prog4.java*/
public class Prog4 {
    public static void main (String args[])
    {
        System.out.println(args[0]);
        System.out.println(args[1]);
    }
}

• If you run this program with the command
  > java Prog4 Dopey Grumpy
• the two strings entered at the command line, “Dopey” and “Grumpy,” are stored in the array args.
• Consequently, args[0] holds the string “Dopey” and args[1] holds “Grumpy.”
• Notice that arrays are indexed from 0, as they are in C++.
• Output:
  Dopey
  Grumpy
Input/Output Trial!

• Write a program that will take one input, teacher’s last name. Then display it back for the user with a welcome message. So, if we run the program with the command:

```
>java tecsProg2 maria
```

The program will display back:

Output:

Hello Maria, Welcome to TecS at Brooklyn College!
Keyboard Input!

• System.in is an InputStream
• Make sure to include:
  – import java.io.*;
• By default Java includes Java.lang package.
• Example:
  ```java
  // keyboard input
  BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));
  String name = keyboard.readLine();
  System.out.println("your name is" + name);
  ```

/*saved as Prog5.java*/
import java.io.*;
public class Prog5 {
public static void main (String args[]) throws Exception {
    // keyboard input
    System.out.println("Type your name");
    BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));
    String name = keyboard.readLine();
    System.out.println("your name is" + name);
}
}
Strings

- String constants: "Carl"
- String variables: String name = "Carl";
- String length: int n = name.length();
Concatenation

- String fname = "Harry";
  String lname = "Hacker";
  String name = fname + lname;
- name is "HarryHacker"
- If one operand of + is a string, the other is converted to a string:
  String a = "Agent";
  String name = a + 7;
- name is "Agent7"
Converting between Strings and Numbers

- **Convert to number:**
  ```java
  int n = Integer.parseInt(str);
  double x = Double.parseDouble(x);
  ```

- **Convert to string:**
  ```java
  String str = "" + n;
  str = Integer.toString(n);
  ```
Substrings

- String greeting = "Clown";
  String sub = greeting.substring(1, 4);

- Supply start and “past the end” position
- First position is at 0
  \[ C_1 \text{lo}_3 \text{w}_4 \text{n} \]
- substring length = “past the end” - start
Reading Input

• String input = JOptionPane.showMessageDialog(prompt)
• Convert strings to numbers if necessary:
  int count = Integer.parseInt(input);
• Conversion throws an exception if user doesn't supply a number--see chapter 15
• Add
  System.exit(0)
to the main method of any program that uses JOptionPane
An Input Dialog

How many nickels do you have?

OK  Cancel
This program tests input from an input dialog.
String input = JOptionPane.showInputDialog("How many nickels do you have?");
int count = Integer.parseInt(input);
myPurse.addNickels(count);

input = JOptionPane.showInputDialog("How many dimes do you have?");
count = Integer.parseInt(input);
myPurse.addDimes(count);

input = JOptionPane.showInputDialog("How many quarters do you have?");
count = Integer.parseInt(input);
myPurse.addQuarters(count);

double totalValue = myPurse.getTotal();
System.out.println("The total is " + totalValue);

System.exit(0);
Syntax Errors vs. Logic Errors

• Syntax errors
  System.ouch.print("...");
  System.out.print("Hello");

• Detected by the compiler

• Logic errors
  System.out.print("Helo");

• Detected (hopefully) through testing
String Concatenation

public class Prog3 //saved as prog3.java
{
    public static void main (String args[])
    {
        System.out.println("Dopey" + 2+3+4);
        System.out.println(2+3+4+"Dopey");
        System.out.println("Dopey" +
                        ( 2+3+4 ));
        System.out.println(2+3+4);
        System.out.println( ( 2+3+4 ));
    }
}

Output:
Dopey234
9Dopey
Dopey9
9
9

Here, you should notice that:

• Addition, as usual, is performed left to right.
• The argument to println is always a String.
• If the argument to println is x+y and either x or y is a string, then + effects string concatenation.
• For example, consider the method call println("Dopey” + 2+3+4). The integer 2,3,4 is converted to a string and the concatenated string “Dopey234” is passed to println.
• However, in the method call println(2+3+4+”Dopey”), the first/Second plus represents addition, the third effects concatenation. So, the string “9Dopey” is passed to the println method.
Selection and Iteration

/*Reads 4 grades from the command line. Thus, the grades are stored as strings and must be converted to a numeric type before any processing can occur.*/

public class Grades1 {
    public static void main( String args[])
    {
        char answer;
        int grade, sum = 0;
        double average;
    /*Iteration Structure*/
        for (int i = 0; i < 4; i++)
            sum = sum + Integer.parseInt(args[i]) ;
    /* converts from string to int*/
        average = sum/4.0;
    /*note that an explicit cast is not necessary*/
        System.out.print("Average is "+ average + " ");
    /*Selection Structure*/
        if (average >= 90)
            System.out.println('A');
        else if (average >= 80)
            System.out.println('B');
        else if (average >= 70)
            System.out.println('C');
        else if (average > 60)
            System.out.println('D');
        else
            System.out.println('F');
    } //end main()
    } //end class Grades1
Grades Program

- To run the Grades1 program:
  
  ```java
  > java Grades1 80 90 70 60
  ```

  Notice that
  - `args[0]` is “80”
  - `args[1]` is “90”
  - `args[2]` is “70”
  - `args[3]` is “60”

- In order to calculate an average, the strings stored in `args` must be converted to integers. Java provides a class `Integer` with a static method, `parseInt(String s)`, that does exactly that -- accepts a string of digits and converts that string to an integer.

- If the string contains any nonnumeric characters, the program will crash.

- Again, notice that `parseInt` is a static method so it can be invoked without instantiating an `Integer` object.

- Finally, while loops, do-while loops and the switch statement work exactly as in C++.