CS5000: Foundations of Programming

Mignon Kang, PhD
Computer Science, Kennesaw State University
Classes

- Most important component of object-oriented programming
- Every program is a class in JAVA
- Every library consists of classes
- Programmer-defined type
- Consists of “Fields” and “Methods”
Primitive type values vs. Class Type values

- A primitive type value is a single piece of data
- A class type value can have multiple data and actions
  - Can have multiple different data
  - Can have methods that represent actions.
Classes

Philosophy tries to understand human problems through human language

Software Engineering tries to solve human problems with computer language
Classes

- Model real world objects
  - State and Actions

For example:
- Human (Name, DOB, Gender, … : Running, Sleeping,…)
- Cars (Model name, Manufacturer, Current speed, Current gear, … : Going forward, Applying brake, Changing gear, …)
- Students (Name, ID, … : Adding/Dropping classes, Applying graduation…)
Classes

- A “template” / “blueprint” that is used to create objects
- Consists of “field”, “static field”, “method”, “static method” and “constructor”
  - Field: hold the state of the class
  - Method: represent the behavior of the class
  - Constructor: create a new instance of the class.
- In JAVA, all classes are inherited from java.lang.Object implicitly.
Objects

- Often use object and instance word interchangeably
- A unique copy of a class
- Share the blueprint of a class, but has unique member data

Here is an example class:

```java
class Human {
    // description of a person goes here...
}
```

Here are some objects of that class:
## Classes

<table>
<thead>
<tr>
<th>Object</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance of a class</td>
<td>Blueprint or template from which objects are created</td>
</tr>
<tr>
<td>Real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.</td>
<td>Group of similar objects.</td>
</tr>
<tr>
<td>Physical entity.</td>
<td>Logical entity.</td>
</tr>
<tr>
<td>Object is created through new keyword mainly e.g. Student s1=new Student();</td>
<td>Declared using class keyword e.g. class Student{}</td>
</tr>
<tr>
<td>Object is created many times as per requirement.</td>
<td>Class is declared once.</td>
</tr>
<tr>
<td>Object allocates memory when it is created.</td>
<td>Class doesn't allocated memory when it is created.</td>
</tr>
<tr>
<td>There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.</td>
<td>There is only one way to define class in java using class keyword.</td>
</tr>
</tbody>
</table>
Define a Class

class Human
{
    // Fields
    public String strName;
    private String strJob;
    protected int nYearOfBirth;
    ...
    // Methods
    public void greet()
    {
        System.out.println(“Hi”);
    }
}
Access Modifiers

- Set access levels for classes, variables, methods, and constructors.
  - Visible to the package (default)
  - Visible to the class only (private)
  - Visible to world (public)
  - Visible to the package and all subclasses (protected)
Create an object of a class

- Use **new** operator

```java
Human objMingonKang;
objMingonKang = new Human();
```

Combined as follows:

```java
Human objMingonKang = new Human();
```
In order to refer to a particular instance variables (fields) or methods, preface it with its object name as follows:

```java
Human objMingonKang;
objMingonKang = new Human();
System.out.print(objMingonKang.strName);
objMingonKang.greet();
```
Methods

- A method that returns a value must specify the type of the value

  ```java
  public typeReturned methodName(paramList)
  ```

- "void" keyword shows that it does not return a value

  ```java
  public void methodName(paramList)
  ```
Methods

- Unlike C/C++, a method cannot exist alone without a class.
- Every functions should be declared within a class method.
Methods

- The body of a method that returns a value must also contain one or more "return" statements.
Local/Global variables

- A variable declared within a method definition is called a local variable
- No Global Variable in JAVA
Overloading

- Two or more methods in the same class can have the same method name with different parameters.
- Called “Overloaded functions”
- Ambiguous method invocations will produce an error in JAVA.
public void PrintValues(int nValue1)
{
    ...
}

class 
{
    public void PrintValues(int nValue1, int nValue2)
    {
        ...
    }
}

class 
{
    public void Print()
    {
        PrintValues(1);
        PrintValues(3, 4);
    }
}
public void Add(int nValue)
{
    ...
}

public void Add(float nValue)
{
    ...
}

public static void main(String[] args)
{
    Add(1);
    Add(3.4);
}
A constructor is a special kind of method that is designed to initialize the instance variables for an object:

```java
public ClassName(anyParameters) { code }
```

- A constructor must have the same name as the class
- A constructor has no type returned, not even `void`
- Constructors are typically overloaded
Constructors

- A constructor is called when an object of the class is created using `new`:

  ```java
  ClassName objectName = new ClassName(anyArgs);
  ```

  - The name of the constructor and its parenthesized list of arguments (if any) must follow the `new` operator.
  - This is the **only** valid way to invoke a constructor: a constructor cannot be invoked like an ordinary method.

- If a constructor is invoked again (using `new`), the first object is discarded and an entirely new object is created.
  - If you need to change the values of instance variables of the object, use mutator methods instead.
public class Students {

    private String strName;
    private int nExamScore;

    Students() {
        strName = "No Name Given";
        nExamScore = 0;
    }

}

Students objStudent = new Students();
If an explicit name for the calling object is needed, the keyword this can be used

- `myInstanceVariable` always means and is always interchangeable with `this.myInstanceVariable`

- `this` must be used if a parameter or other local variable with the same name is used in the method

  Otherwise, all instances of the variable name will be interpreted as local

  ```java
  int someVariable = this.someVariable
  ```
public class Students
{
    private String strName;
    private int nExamScore;

    Students(String strName, int nExamScore)
    {
        this.strName = strName;
        this.nExamScore = nExamScore;
    }
}

Students objStudent = new Students(“Min”, 100);
Destructor?

- Nope.
- Java is a garbage collected language
- Java deallocates every memory space automatically.
Setter/Getter methods

- To make the class fields accessible, you need to add setter and getter methods for that state.
- Hide instance variables, and provide public methods that retrieve/change the variables.
public class Printer {

    private String name;

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

    public String getName() {
        return name;
    }

}
A static variable/method is one that's associated with a class, not objects of that class.

```java
public class Stuff {
    public static String name = "I'm a static variable";
}

public class Application {
    public static void main(String[] args) {
        System.out.println(Stuff.name);
    }
}
```
Display 4.6  A Formal Parameter Used as a Local Variable

1  import java.util.Scanner;

2  public class Bill
3  {
4    public static double RATE = 150.00; //Dollars per quarter hour

5    private int hours;
6    private int minutes;
7    private double fee;

(continued)
Display 4.6  A Formal Parameter Used as a Local Variable

```java
public void inputTimeWorked()
{
    System.out.println("Enter number of full hours worked");
    System.out.println("followed by number of minutes:");
    Scanner keyboard = new Scanner(System.in);
    hours = keyboard.nextInt();
    minutes = keyboard.nextInt();
}
```

```java
public double computeFee(int hoursWorked, int minutesWorked)
{
    minutesWorked = hoursWorked*60 + minutesWorked;
    int quarterHours = minutesWorked/15;  // Any remaining fraction of a
    // quarter hour is not charged for.
    return quarterHours*RATE;
}
```

```java
public void updateFee()
{
    fee = computeFee(hours, minutes);
}
```

(continued)
A Formal Parameter Used as a Local Variable

```java
public void outputBill()
{
    System.out.println("Time worked: ");
    System.out.println(hours + " hours and " + minutes + " minutes");
    System.out.println("Rate: $" + RATE + " per quarter hour.");
    System.out.println("Amount due: $" + fee);
}
```
Display 4.6  A Formal Parameter Used as a Local Variable

```java
1   public class BillingDialog
2   {
3       public static void main(String[] args)
4           {
5               System.out.println("Welcome to the law offices of");
6               System.out.println("Dewey, Cheatham, and Howe.");
7               Bill yourBill = new Bill();
8               yourBill.inputTimeWorked();
9               yourBill.updateFee();
10              yourBill.outputBill();
11              System.out.println("We have placed a lien on your house.");
12              System.out.println("It has been our pleasure to serve you.");
13           }
14   }
```

This is the file BillingDialog.java.
Display 4.6  A Formal Parameter Used as a Local Variable

SAMPLE DIALOGUE

Welcome to the law offices of
Dewey, Cheatham, and Howe.
Enter number of full hours worked
followed by number of minutes:
3 48
Time worked:
2 hours and 48 minutes
Rate: $150.0 per quarter hour.
Amount due: $2250.0
We have placed a lien on your house.
It has been our pleasure to serve you.