7. Class, OOP (2), file in&out, data structure
August 23, 2015

Minhaeng Lee
FOP and OOP

• **Function Oriented Programming**
  – Function call from main
  – Main – function – main – function ...
  – Difficult to management
  – To replace function, we need to edit file itself
  – Pure C style. Nowadays, not popular.

• **Object Oriented Programming**
  – Object call from main
  – Main – object – method – object – main ...
  – Easier to management
  – To replace class, we need to change file (class)
  – Still popular in Java, C++, python, javascript, ....
Function Oriented Programming

Class concept doesn’t have any big meaning.
Class

• Consist of
  – Variables
    • Variables declared in class (but outside of method)
  – Methods
    • Similar meaning with function. We use both
  – Constructor
    • A Method called when a class is initialized (=allocated to memory by new keyword)

• Instance?
  – The class variable made by new keyword
  – Memory allocated class variable
  – Example
    • a is an instance
    • MyClass a = new MyClass();

• Inheritance (today)
• Polymorphism (next week)
Constructor

• A method called at memory allocation
• Initialization process involved
  – Has class name as its name
  – Similar to function but without return type
    – `public <className> (<parameters>) { }`

• Default constructor is defined already
  – `public <className>() { }`
```java
public class Circle {
    // Instant variables
    int x, y;
    private int radius;
    private double area;

    // Constructor
    public Circle(int x, int y) {
        // this refers to in-class variable
        this.x = x;
        this.y = y;
    }

    // Get and Set
    public void setRadius(int radius) {
        this.radius = radius;
        this.area = Math.PI * 2 * radius;
    }

    public double getArea() {
        return area;
    }

    @Override
    public String toString() {
        return "x:" + x +", y:" + y +", radius:" + radius +", area:" + area;
    }
}
```

```java
public class ClassTest {
    public static void main(String[] args) {
        // Allocate instance
        Circle circle = new Circle(10, 10);
        circle.setRadius(5);
        circle.radius = 10;
        System.out.println(circle.getArea());
        System.out.println(circle.toString());
    }
}
```
Access Specifier

• **Public**
  – Allow to everybody
  – Outside of class to local variables/methods

• **private (default)**
  – Allow to class itself
  – Inside of class to its local variables/methods

• **Protected**
  – Allow to its descent
  – Inside of descent (children inherit itself) to class itself
extends keyword for inheritance

- Sub has all properties (variables and methods) of Super
- When?
  - Divide file (or class) based on their role
  - Make common variables or methods of Sub into Super

```java
class Super{
    ....
    ....
}

class Sub extends Super{
    ....
    ....
}
```

http://www.tutorialspoint.com/java/java_inheritance.htm
extends keyword for inheritance

class Calculation{
    int z;
    public void addition(int x, int y){
        z=x+y;
        System.out.println("The sum of the given numbers:"+z);
    }
    public void Subtraction(int x,int y){
        z=x-y;
        System.out.println("The difference between the given numbers:"+z);
    }
}

public class My_Calculation extends Calculation{
    int z;
    public void multiplication(int x, int y){
        z=x*y;
        System.out.println("The product of the given numbers:"+z);
    }
    public static void main(String args[]){
        int a=20, b=10;
        My_Calculation demo = new My_Calculation();
        demo.addition(a, b);
        demo.Subtraction(a, b);
        demo.multiplication(a, b);
    }
}
extends keyword for inheritance

http://www.tutorialspoint.com/java/java_inheritance.htm
Practice: Programmer Company

- Senior Programmer
  - Team Members

- Programmer
  - Computer ID

- Employee
  - Name
  - Age
public class Employee {
    // Name
    String name;
    // Age
    int age;

    public Employee(String name, int age) {
        // This refers to local variable.
        // Without "this" keyword, it refers parameter
        this.name = name;
        this.age = age;
    }

    // toString method to print out its info.
    public String toString() {
        return "my name is \"+name+\", and \"+age+\" years old.\";
    }
}
public class Programmer extends Employee{
    // Programmer has its own computer.
    String computerID;

    public Programmer(String name, int age, String computerID) {
        super(name, age);
        this.computerID = computerID;
    }

    public String toString()
    {
        return super.toString() + "\n" +
                "My computer ID is " + computerID;
    }
}
import java.util.Vector;

public class SeniorProgrammer extends Programmer{
    Vector<Programmer> teamMembers;

    public SeniorProgrammer(String name, int age, String computerID) {
        super(name, age, computerID);
        teamMembers = new Vector<Programmer>();
    }

    //Add member function
    public void addMember(Programming programmer){
        this.teamMembers.add(programmer);
    }

    public String toString(){
        return super.toString()+"\n\n "+
        "Our team information \n "+teamMembers.toString();
    }
}
public class Company {
    public static void main(String[] args) {
        Employee emp = new Employee("jack", 25);
        Programmer prog = new Programmer("jim", 25, "ABCDE");
        SeniorProgrammer boss = new SeniorProgrammer("boss", 40, "ABCDE123");
        System.out.println(emp);
        System.out.println(prog);
        System.out.println(boss);
        boss.addMember(prog);
        System.out.println(boss);
    }
}
Overloading and overriding

• Overriding
  – Define different method with same name with different parameters in a class
    • Add(int a, int b);
    • Add(String a, String b);

• Overriding
  – Re-define ancestor’s methods in descendant class
    – Add(int a, int b); //In ancestor
    – Add(int a, int b); //In descendant
Polymorphism

- **IS-A** relationship
- Descendant IS-A Ancestor in OOP concept

**Example**
- A Deer IS-A Animal
- Subclass IS-A Superclass
- You IS-A Human

**If we have class Dog and Animal, and Dog IS-A Animal then**
- We can set Dog as Animal
  - `Animal barkAnimal = new Dog();`
  - `Animal mewAnimal = new Cat();`
Virtual Methods

• Animal barkAnimal = new Dog();
• Animal mewAnimal = new Cat();
• In Dog, and Cat class, they have their own feed() methods which is overridden ones.

• Even if some descendants are called as their ancestor, when their methods are called, they execute their own overridden ones. Those run in different way.
  – barkAnimal.feed();
  – mewAnimal.feed();
Read code!
Data Structure: preview

- **Stack**
  - First in, last out
- **Queue**
  - First in, First out (FIFO)
- **List**
  - Versatile list typed data structure
- **Tree**
  - Node connected to its children
  - Binary tree: tree has at most two children
- **Map**
  - Also called hash map
  - Data mapped via hash function
  - Can access elements via “Object” called key
Stack

• First in – last out
• `push()` and `pop()` method
• Useful to solve searching problem
  – Think, when you got deadend, then?

Let’s run example code!
Queue

- First in, First out
- Useful for time work scheduling
List

• Super class : List
  - List a = new Vector();

• Example class : Vector
  - Let’s run example code (Uploaded in our website)

• Useful for various purposes
Tree

- Stems from Root
- Each element is called node
- Each node may have its own children
- Useful for data search, sorting

Figure: Tree data structure
Map

- Keys are connected to their data
- Hash function converts keys into index
- Well-formed hash function does not allow index duplication

Let’s run example code!
File read/write

- How to read and write file
- Useful to save data
- Useful to use pre-defined parameter

- **PrintWriter**
  - When we write file

- **BufferedReader**
  - When we read file
package fileinout;
import java.io.FileWriter;

public class PrintWriterDemo {
    public static void main(String[] args) throws Exception {
        String filename = "fileName.txt";
        String[] linesToWrite = new String[] { "abc", "bddd" };
        boolean appendToFile = true;

        PrintWriter pw = null;
        if (appendToFile) {
            pw = new PrintWriter(new FileWriter(filename, true));
        } else {
            pw = new PrintWriter(new FileWriter(filename));
        }
        for (int i = 0; i < linesToWrite.length; i++) {
            //Use like System.out.println
            pw.println(linesToWrite[i]);
        }
        pw.flush();
        pw.close();
    }
}
Buffered Reader – type it!

```java
package fileinout;

import java.io.BufferedReader;
import java.io.FileReader;

public class BufferedReaderDemo {
    public static void main(String[] args) throws Exception {
        String thisLine = null;
        try{
            // open input stream test.txt for reading purpose.
            FileReader fr = new FileReader("fileName.txt");
            BufferedReader br = new BufferedReader(fr);
            while ((thisLine = br.readLine()) != null) {
                System.out.println(thisLine);
            }
            br.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```
Where is the written file?
Questions
References

• IBM