CONTENTS

Basic JAVA structure

Object – Oriented Programming
Codecademy : JAVA
Codecademy

- Useful tutorial page for learning programming languages.
- Sign in and learn JAVA course.
- Java course is consist of 4 chapters: Introduction, Control statements, Object-Oriented, Data structure.
Basic JAVA code structure
Main Function

```java
public class SoftExperience3 {
    public static void main(String[] args) {
        /*
         * This is main function.
         */
        System.out.println("Hello World!!");
        SoftExperience3 se3 = new SoftExperience3();
        int addResult = se3.add(10, 20);
        System.out.println(addResult);
        return;
    }

    public int add(int x, int y) {
        System.out.println("Addition " + x + " and " + y);
        return x + y;
    }
}
```
Input & Output

```java
import java.util.Scanner;

public class SoftExperience3 {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        System.out.print("Enter a string : ");
        String sin = in.nextLine();

        System.out.print("Enter a integer : ");
        int iin = in.nextInt();

        System.out.print("Enter a double : ");
        double din = in.nextDouble();

        in.close();

        System.out.println("Your inputs are " + sin + ", " + iin + ", " + din);

        return;
    }
}
```

Enter a string : Hello World!
Enter a integer : 2147483647
Enter a double : 3.141592
Your inputs are Hello World!, 2147483647, 3.141592
Basic operations

- Basic operations of JAVA is same to C or C++.

- Data types
- Arithmetic operations
- Relation & logical operations
- Selection statements
- Loop statements
- Concepts of blocks

- So, we will skip them.
Object-Oriented Programming
Object-Oriented Programming

- **Java** can design classes, objects, and methods that can perform certain actions.
- **Java** is a collection of classes. (Object class)
  - **Class** is a set of instructions that describe how a data structure should behave.
  - **Instance** is a real data that is created from class.
- Distinct **class** and **instance**.
public class Point {
    /* Member variable */
    private int x, y;

    /* Constructor */
    public Point(int x, int y){
        this.x = x;
        this.y = y;
    }

    /* Method */
    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void print(){
        System.out.println("Point position : (" + x + ", " + y + ")");
    }
}

public class SoftExperience3 {

    public static void main(String[] args) {
        Point p = new Point(10, 20);
        p.print();
        return;
    }
}
Static

• Normal variables are made when each instances are created. But, static variables is included in the class alone.

• All instance share the static variables.

• Static variables can be used for communication among the instances.

• Final variables are constant value.

• Final variables cannot be modified.

• Qualifiers are same to C++.

---

Final

Qualifier

---
public class Point {
    /* Member variable */
    private int x, y;
    private int id;

    public static int pointId;

    public final String TAG = "Point";

    /* Constructor */
    public Point(int x, int y){
        this.x = x;
        this.y = y;
        this.id = pointId++;
    }

    /* Method */
    public int getId(){
        return id;
    }
}

public class SoftExperience3 {

    public static void main(String[] args) {
        Point p1 = new Point(10, 20);
        Point p2 = new Point(4, 8);
        Point p3 = new Point(13, 5);

        System.out.println("Point 1 id : " + p1.getId());
        System.out.println("Point 2 id : " + p2.getId());
        System.out.println("Point 3 id : " + p3.getId());

        System.out.println("The number of " + Point.TAG + " : " + Point.pointId);
        return;
    }
}
Overloading

this

- Constructor and methods can be overloaded.
- With different parameters, constructor and methods can have same name.
- Same name is able to be an advantageous in programming.
- “this” means current ‘instance’.
- With “this”, we can call same name of variables and methods also.
Overloading

this

```java
public class SoftExperience3 {
    public static void main(String[] args) {
        Point p1 = new Point(10, 20);
        Point p2 = new Point(20, 10);
        Point p3 = new Point();

        p1.multiply(2);
        p2.multiply(2.5f);

        return;
    }
}
```

```java
public class Point {
    /* Member variable */
    private int x, y;
    private int id;

    public static int pointId;

    /* Constructor */
    public Point(){
        this(0, 0);
        this.id = pointId++;
    }

    public Point(int x, int y){
        this.x = x;
        this.y = y;
        this.id = pointId++;
    }

    /* Method */
    public void multiply(int m){
        x = x * m;
        y = y * m;
    }

    public void multiply(float m){
        x = (int)(x * m);
        y = (int)(y * m);
    }
}
```
Inheritance

- **Inheritance** provide recycling of module and briefness of code.

- We can implement hierarchy of class with **inheritance**.

- **Inheritance** can express "is-a" relation or "has-a" relation.

- All classes of Java has top-level class, named java.lang.Object.
Inheritance

```java
class SoftExperience3 {
    public static void main(String[] args) {
        Circle c1 = new Circle(0, 0, 5);
        c1.calculateArea();
        System.out.println(c1.getArea());
        return;
    }
}

class Circle extends Figure {
    private Point center;
    private int radius;

    public Circle(int x, int y, int radius) {
        this.center = new Point(x, y);
        this.radius = radius;
    }

    public Circle(Point center, int radius) {
        this.center = center;
        this.radius = radius;
    }

    public void calculateArea() {
        this.area = radius * Math.PI;
    }
}
```
• “super” means parent class. It indicate not only parent, but also ancestors.

• By “super”, we can use parent’s method or constructors.

• In child class, we can override parent’s methods or constructors with additional functions.

• Abstract class implements only concepts of abstract methods.

• Child class should override abstract methods of its parent class.

• When a method is dependent to its child’s properties...
public abstract class Triangle {
    private double line1;
    private double line2;
    private double line3;

    public Triangle(double line1, double line2, double line3) {
        this.line1 = line1;
        this.line2 = line2;
        this.line3 = line3;
    }

    public double getEdgeLength(){
        return line1 + line2 + line3;
    }

    public void print(){
        System.out.println("I am a triangle");
    }

    public abstract double getArea();
}
Super
Overriding
Abstract

```java
public class RegularTriangle extends Triangle{
    private double line;

    public RegularTriangle(double line){
        super(line, line, line);
        this.line = line;
    }

    public void print(){
        super.print();
        System.out.println("I am a regular triangle");
    }

    @Override
    public double getArea() {
        return Math.sqrt(3) / 4 * line;
    }
}
```
public class IsoscelesTriangle extends Triangle{

    private double width;
    private double height;

    public IsoscelesTriangle(double width, double height){
        super(Math.sqrt(Math.pow(width/2, 2) + Math.pow(height, 2)),
             Math.sqrt(Math.pow(width/2, 2) + Math.pow(height, 2)),
             width);
        this.width = width;
        this.height = height;
    }

    public void print(){
        super.print();
        System.out.println("I am a isosceles triangle");
    }

    @Override
    public double getArea() {
        return width * height * 0.5;
    }
}
• **Interface** has only **abstract** methods.

**interface**

• We can implements multiple inheritance with **interfaces**.

• A class that is created from **interface** should implements the **abstract** methods of **interface**.

• **Interface** itself also can be extended from other interfaces.
Interface

```java
public interface OnClickListener {
    public abstract void onClick();
}

public interface OnLongClickListener {
    public abstract void onLongClick();
}

public class Button implements OnClickListener, OnLongClickListener {
    public Button() {
    }

    public void print() {
        System.out.println("I am a button");
    }

    @Override
    public void onClick() {
        System.out.println("Button is clicked!");
    }

    @Override
    public void onLongClick() {
        System.out.println("Button is long clicked!");
    }
}

public class SoftExperience3 {
    public static void main(String[] args) {
        Button btn = new Button();
        btn.onClick();
        btn.onLongClick();
        return;
    }
}
```

Button is clicked!
Button is long clicked!
Callback with interface

- Let’s see how the interface is used in **Android** platform.

- By interface, we can make **callback** functions.

- **Callback** is any code that is passed as an argument to other code, which is expected to callback the argument at a given time.

- We can design different instructions at each form and at each time.

- Each objects can have different method.
public class TextView {
    private OnClickListener mOnClickListener;
    private OnLongClickListener mOnLongClickListener;

    public TextView(){}

    public void print(){
        System.out.println("I am a textview");
    }

    public void setOnClickListener
            (OnClickListener mOnClickListener){
        this.mOnClickListener = mOnClickListener;
    }

    public void setOnLongClickListener
            (OnLongClickListener mOnLongClickListener){
        this.mOnLongClickListener = mOnLongClickListener;
    }

    public void onClick(){
        if( mOnClickListener != null )
            mOnClickListener.onClick();
    }

    public void onLongClick(){
        if( mOnLongClickListener != null ){
            mOnLongClickListener.onLongClick();
        }
    }
}
public class SoftExperience3 {

public static void main(String[] args) {
    final int temp = 0;

    TextView tv = new TextView();
    tv.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick() {
            System.out.println("Hello!!!");
        }
    });

    tv.setOnLongClickListener(new OnLongClickListener() {
        @Override
        public void onLongClick() {
            System.out.println("Temp = " + temp);
        }
    });

    tv.setOnClickListener();
    tv.setOnLongClickListener();

    return;
}
}
THANK YOU!!!