Java OOP 编程基本概念

1、什么是类，对象，实例?

类：描述具有一组相同的属性与操作的实体

```
public class Course{
    private Sting cno, cname;
    private int cxf;
    private String ctype;

    ...
    public void addCource(.......) {
    ......
    }

    public Courrce queryCourseByCno(String cno) {
    ...............
    }
}
```

对象：变量：用来存储实例的地址，引用某个实例。

```
Cource c1, c2;  // c1, c2对象
```

实例：Instance

```
c1 = new Cource();
c2 = c1;
Cource c3 = new Cource();

New作用：1、给实例分配内存空间（属性需要空间，函数需要入口地址）
         2、返回内存空间的首地址
```

2、Java对象，实例是如何管理?

```
public static void main(String[] args) {
    int x = 0;
    Cource c1 = new Cource();
    for (int i = 0; i < = 100; i++) {
    ....
    }
}
```

对象和变量通过栈空间存储，生命周期依照栈的次序 "后进先出"

3、什么是构造函数?

1)、为什么要用构造函数

2)、 构造函数有什么特点

与类同名，没有任何返回值类型（void 都不能有）；一般为public

```
public class Course{
    private Sting cno, cname;
    private int cxf;
    private String ctype;

    public Cource() {
    }
    public Cource(String cno, String cname, int cxf, String
 type) {
        this.cno = cno;
        this.cname = cname;
        ..........
    }

    ...
    public void addCource(.......) {
    ......
    }

    public Courrce queryCourseByCno(String cno) {
    ...............
    }
}
```

3)、目的和作用

```
Cource c1 = new Cource();
Cource c2 = new Cource("100010", "高数", 6, "必修课");
```

4)、如何调用?

构造函数为系统在实例化过程中系统首先会自动调用自动的构造函数

如果没有写构造函数，系统编译时调用默认的构造函数，默认的构造函数为了实现语法的完整性，

4属性构造器函数(get-set函数)

```
public class Course{
    private Sting cno, cname;
    private int cxf;
    private String ctype;

    public Cource() {
    }
    public Cource(String cno, String cname, int cxf, String
 type) {
        this.cno = cno;
        this.cname = cname;
        ..........
    }
    // get函数
    public String getCno() {
        return cno;
    }
    //set函数
    public void setCno(String cno) {
        this.cno = cno;
    }
}
```

5 static, final

Static

1)静态函数, 在编译后和运行前已经有这个方法的入口地址

```
public class A {
    public static void fun1() {

    }
    public void fun2() {

    }

A.fun1();//类函数
Math.random();
A.fun2();//error
A obj = new A();
obj.fun2();//实例函数
```

2)静态属性域变量

```
public class A {
    static int x = 0;
    int y = 0;
    public A() {
        x ++;
        y ++;
    }
    public void disp() {
        System.out.println("x=" + x + " y=" + y);
    }
}


A obj1 = new A();
obj1.disp();
A obj2 = new A();
obj2.disp();
```

Final

1)final 变量

```
final double pi = 3.14159;
```

2)final 类

```
public final class  MyString {
    ...
}

MyString 不能派生子类

public class A extend MyString { // error
    ....
}

Math, String 等为final类
```

3)final函数(can't be overwrite)

```
public class A {
    public final vid fun() {
        ...
    }
}

public class B extends A {
    public void fun() { //error
        ...
    }
}
```

6 继承extends

```
public class MyLogin extends JFrame {

}
```

1)为什么需要继承?

代码的复用性

2)继承的本质

```
public  class A {
    private int x;
    public void funA() {
        ...
    }
}

public class B extends A {
    public void funB() {

    }
}
B obj = new B(); //子类实例化，系统首先自动将父类实例化。
obj.funA();
obj.funB();
子类调用父类的公共的方法和属性通过子类本身的指向父类公有方法和属性的的指针调用
```

3)继承了什么?

public , protected private

7、继承下的构造函数

子类实例化，系统首先自动将父类实例化。首先自动调用父类的构造函数， 然后是子类本身的构造函数。

```
class Person {
    private Strign pid, pname;
    public Person() {
    }
    public Person(String pid, String pname) {
        this.pid = pid;
        this.pname = pname;
    }
}

public class  Student extends Person {
    private String smajor;
    public Student() {
    }
    public Student (String pid,  String pname, String smajo
r) {
        super(pid, pname); //必须放在第一条
        this.smajor = smajor;
    }
}
```

8、多态

```
public double area();
```

1) abstract

```
public abstract class Shape {
    public final double pi = 3.14;
    public abstract double area();
    public void dispArea() {
        System.out.println("area=" + area());
    }
}

public class Circle extends Shape {
    private double r;
    @Override  //上传并覆盖父类函数,将原先父类的函数重新覆盖
    public double area() {
        return pi * r * r;
    }
}

Circle obj = new Circle(5.0);
obj.dispArea();

Shape obj2 = new Circle(10.0);
obj2.area();

Shape obj3 = new Shape(); // error


public class Rect extends Shape {
    double w,  h;
    .......
    public double area() {
        return w * h;
    }
}

Shape shape = new Circle(10.0);
shape.area();
shape = new Rect(3, 4);
shape.area();
```

2) 接口

接口中所有的函数都是抽象的

```
public interface Shape {
    double area();
    void dispArea();
}


public class Circle implements Shape {
    private double r;

    @Override
    public double area() {
        return pi * r * r;
    }

    @Override
    public double dispArea() {
        ......
    }
}


public interface StudentDao {
    public void addStudent(Stundent s);
    public void deleteStudent(String sno);

}

public class StudentDaoImpl implements StudentDao {
    public void addStudent(Student s) {
        ......
    }
    public void deleteStudent(String sno) {
        ......
    }
}
```

9、类实例

```
public class Point {
    private int x, y;
    public Point() {}
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int  getX() {
        return x;
    }

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

    public int getY() {
        return y;
    }

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

    public String toString() {
        return  "(" + x + ", " + y + ")";
    }

    public double distanceFromZero() {
        return Math.sqrt(x * x + y *y);
    }

    public double distance(Point  point) {
        return Math.sqrt(
        (this.getX() - point.getX()) * (this.getX() - point
.getX()) +
        (this.getY() - point.getY()) * (this.getY() - point
.getY()));
    }

    public void move(int dx, int dy) {
        this.x = this.x + dx;
        this.y = this.y + dy;
    }

    public Point moveToNewPoint (int dx, int dy) {
        Point p = new Point();
        p.setX(this.x + dx);
        p.setY(this.y + dy);
        return p;
    }
}

public class Test {
    public static void main(String[] args) {
        Point p1 = new Point(10, 15);
        Point p2 = new Point();
        p2.setX(20);
        p2.setX(30);
        System.out.println(p1 + " " + p2);
        p1.move(10, -5);
        System.out.println(p1 + " " + p2);
        Point p3 = p1.moveToNewPoint(10, 20);
        double d1 = p1.distance(p2);
        ......
    }
}
```