# 一、填空

1. 使用 static 修饰符定义的类成员，可以通过类直接访问而不需要创建对象后再访问。
2. 用 abstract 修饰符定义的方法，没有方法体, 使用 abstract 修饰符定义的类不能实例化。
3. 类的中一个成员是一个类的对象时，如果该成员没有被初始化，则该对象的初始值是 null。
4. 在子类构造函数中使用 super 关键字来调用父类的构造函数。
5. Java接口中可以声明 属性常量 和 抽象方法。
6. 用final关键字修饰一个类表示 该类不能被继承。
7. 在子类的实例方法m中要调用父类被覆盖的实例方法m（方法m不带参数且没有返回值）的语句是 `super.m()`。
8. 如有以下类的定义：

```
abstract class A {
     public void fa () {};
     public abstract void fb();
     public abstract void fc();
}

interface I {
     void fx();
}

abstract class B extends A {
     public void fb() {};
     public abstract void fd();
}

public class C extends B implements I {
     …
}
```

则在在class C中必须要实现的方法为 fc fd fx 。

9. 如有下列接口和类的定义：

```
interface I1{  }
interface I2 extends I1{ }
class A implements I2{ }
class B extends A{ }
```

则B类的一个实例对象o的类型可以是 Object B A I2 I1。

10. 下列程序的输出结果是 three two one 1 2 。

```
class C {
     int x;
     String y;
     public C() {
```

```
            this(“1”);
            System.out.print(“one ”);
      }
      public C(String y) {
            this(1, “2”);
            System.out.print(“two ”);
      }
      public C(int x, String y) {
            this.x = x;
            this.y = y;
            System.out.print(“three ”);
      }
      public static void main(String[] args) {
            C c = new C();
            System.out.println(c.x + “ ” + c.y);
      }

}
```

11. 在Java中对于程序可能出现的必检异常，要么用try...catch语句捕获并处理它，要么使用 throws 语句抛出它，由上一级调用者来处理。

12. 在Java中异常分为必检异常和非必检异常二种类型，其中表达式10/0会抛出 非必检异常 类型异常，打开一个不存在的文件会抛出 必检异常 类型异常，通过空引用调用实例方法会抛出 非必检异常 类型异常, 数组越界访问会抛出 非必检异常 类型异常，用throw语句抛出一个自定义的Exception子类异常是 必检异常 类型异常。

## 二、选择题

1. 接口中的成员变量被隐含地声明为 (A)。

A. public static final　　B. public final

C. public static　　D. public abstract

2. 下列叙述中正确的是 (B)。

A. Java中一个类可以有多个直接父类，可以实现多个接口

B. Java中一个类只可以有一个直接父类，可以实现多个接口

C. Java中一个类只可以有一个直接父类，只可以实现一个接口

D. Java中一个类可以有多个直接父类，只可以实现一个接口

3. 关于子类覆盖（Override）父类的方法，下列说法正确的是 (B)。

A．子类方法与父类方法形式参数个数或者类型不同

B．子类方法与父类方法的形式参数个数、形参类型、返回类型相容

C．子类方法与父类方法的访问权限必须相同

D．子类方法与父类方法形式参数名称必须相同

4. 定义类时不能使用的修饰符是 (D)。

A． abstract　　B．final　　C．public　　D．abstract final

5. 下面程序运行后的输出结果为 (D)。静态方法中只能访问静态变量!!!

```
class B{
     int y=3;
     static void showy( ){System.out.println(“y=”+y); }
}

class TestB{
     public static void main(String aaa []){
          B a1=new B( );
          a1.y++;
          a1.showy( );
     }
}
```

A. y=3;　　B. y=4;

C. y=5;　　D. 程序编译出错

6. 给定以下类的定义

```
public class A {
    public A() {
        System.out.println("Constructor");
    }
    public static int i = 0;
}
```

则下列语句中会在控制台中打印出字符串Constructor的是 (C)。

A. A a = null;　　B. int k = A.i;

C. int k = new A().i;　　D. A[] array = new A[1];

7. class A extends B implements C，假定A和B有缺省构造方法，则下面的语句编译和运行正确的是 (C)。

A．A a = new A( ); B b = a; C c = b;

B．B b = new B( ); A a = (A) b;

C．A a = new A( ); B b = a; C c1 = a ,c2 = new A( );

D．A a = new A( ); C c = new A( ); B b = new C( );

8. 给定下列程序，程序的输出结果为 (A)。Base类中没有默认无参构造，必须在Derived中显示调用super(s)

```
class Base {
     public Base(String s) {
          System.out.print("B");
     }
}

public class Derived extends Base {
     public Derived (String s) {
          System.out.print("D");
     }
     public static void main(String [] args) {
          new Derived ("C");
     }

}
```

那么结果为?

A. 编译错误　　B. DB　　C. BD　　D. BDC

9. 已知如下目录结构（dira和dirb为目录）

dira

|---A.class

|---dirb

　　|---B.class

和如下源代码：

```
import dira.*;
class C {
      A a;
      B b;
}
```

那么要使源代码通过编译，需要在源代码中添加 (C)。

A. package dira;

B. package dirb;

C. package dira.dirb;

D. package dirb.dira;

10. 给定下列程序

```
interface I { }
class A implements I { }
class B extends A { }
class C extends B {
     public static void main(String[] args) {
          B b = new B();
          ____________
     }
}
```

在横线处添加下面哪条语句运行时会产生异常 (C)。ClassCastException

A. A a = b;

B. I i = b;

C. C c = (C) b;

D. B d = (B) (A) b;

11. 下面程序的输出结果是 (C)。

```
class C {
     public static void main(String[] args) {
          try {
              System.out.print(10 + 10 / 0);
          } catch(NullPointerException e1) {
              System.out.print("a");
          } catch(RuntimeException e2) {
              System.out.print("b");
          } finally {
              System.out.print("c");
          }
     }
}
```

A. a　　B. ac　　C. bc　　D. abc

12. 下面程序的执行结果是 (D)。

```
public class MyProgram{
      public static void main (String args[]){
        try{
         System.out.print("Hello Java.");
           }
        finally{
              System.out.print("Finally Java.");
           }
   }
}
```

A． 无法编译，因为没有catch子句　　B． Hello Java.

C． Finally Java.　　D. Hello Java. Finally Java.

13. 下面程序的执行结果是 (C)。

```
public class HomeWorkTest {
    public static void main(String[] args) {
        new B().display();
    }
}

class A {
    public void draw() {
        System.out.print("Draw A.");
    }

    public void display() {
        draw();
        System.out.print("Display A.");
    }

}

class B extends A {
    public void draw() {
        System.out.print("Draw B.");
    }

    public void display() {
        super.display();
        System.out.print("Display B.");
    }
}
```

A. Draw A.Display A.Display B.　　B. Draw A.Display B.Display A.

C. Draw B.Display A.Display B.　　D. Draw B.Display B.Display A.

14. 语句 int[]  m = new int[5];则 m[5]=10;会有 (C)。

A. 编译运行都正确;

B. 编译不正确

C. 会引发ArrayIndexOutOfBoundsException异常

D. 会引发NullPointerException异常

15. 下面程序执行结果是 (D)。

```
public class HomeWorkTest {
    public static void main(String args[]) {
        try {
            System.out.print("try.");
            return;
        } catch (Exception e) {
            System.out.print("catch.");
        } finally {
            System.out.print("finally.");
        }
    }
}
```

A. try.catch.finally.　　B. try.

C. try.catch.　　D. try.finally

16. 给定下列程序，下面说法正确的是 (B)。

```
public class HomeWorkTest {
    public void m1() throws IOException {
        try {
            throw new IOException();
        } catch (IOException e) {

        }

    }
    public void m2() {
        m1();               // 此处编译报错,未处理异常
    }
}
```

A. 因m1方法里已经捕获了异常，因此m2里调用m1()时不用处理异常，程序编译通过

B. m2或者用throws声明异常，或者在方法体里面用try/catch块去调用m1并捕获异常，否则编译报错

C. m2方法体里面必须用try/catch块去调用m1并捕获异常，否则编译报错

D. m2方法必须用throws声明异常，否则编译报错

17. 给定下列程序，下面说法正确的是 (A)。因为RuntimeException不是必检异常

```
public class HomeWorkTest {
    public void m1() throws RuntimeException {
        throw new RuntimeException();
    }
    public void m2() {
        m1();
    }
}
```

A. 程序编译通过

B. m2或者用throws声明异常，或者在方法体里面用try/catch块去调用m1并捕获异常，否则编译报错

C. m2方法体里面必须用try/catch块去调用m1并捕获异常，否则编译报错

D. m2方法必须用throws声明异常，否则编译报错

## 三、判断对错题

1. 包含有抽象方法的类必须是抽象类，抽象类也必须包含有抽象方法。(F) 抽象类不一定要包含抽象方法
2. 一个接口只能继承一个直接父接口。（F） java中的接口支持多继承。类不支持
3. 非抽象类的子类不能是抽象类。（F）非抽象类的子类可以是抽象类
4. 接口类型的引用变量能直接指向一个实现了该接口的类的实例而无需强制类型转换。(T)
5. import语句通常出现在package语句之前。(F)
6. 抽象类中不能定义构造方法。(F) 可以定义抽象方法
7. this关键字可以在类的所有方法里使用。(F) 静态方法中不能用this
8. 类A的所有实例方法都可以在A的子类中进行覆盖(Override)。(F) 被final修饰的方法不能被覆盖
9. 在类的静态初始化块里可以初始化类的静态数据成员和实例数据成员。(F) 不能初始化实例数据成员
10. 由于抽象类不能被实例化，因此方法的参数类型和返回类型都不能是抽象类类型。(F)

## 四、阅读题

阅读程序写出结果

```
class SuperClass{

    static int i = 10;

    static{

        System.out.println(" static in SuperClass");

    }

    {

        System.out.println("SupuerClass is called");

    }

}

class SubClass extends SuperClass{

    static int i = 15;

    static{

        System.out.println(" static in SubClass");

    }
```

```
    SubClass( ){

        System.out.println("SubClass is called");

       }
       public static void main(String[] args){
        int i = SubClass.i;
        new SubClass( );
        new SuperClass( );

    }

}
```

输出结果：

```
static in SuperClass
static in SubClass
SupuerClass is called
SubClass is called
SupuerClass is called
```

阅读程序写出指定语句输出结果，并解释原因

```
class Test5 {
    public static void main(String... args){
        C o1 = new D();
        o1.m(1,1);              //①
        o1.m(1.0,1.0);          //②
        o1.m(1.0f, 1.0f);       //③

        D o2 = new D();
        o2.m(1,1);              //④
        o2.m(1.0,1.0);          //⑤
        o2.m(1.0f, 1.0f);       //⑥
    }
}

class C{
    public void m(int x, int y) {
        System.out.println("C's m(int,int)");
    }
    public void m(double x, double y) {
        System.out.println("C' m(double,double)");
    }
}

class D extends C{
    public void m(float x, float y) {
        System.out.println("D's m(float,float)");
    }
    public void m(int x, int y)  {
        System.out.println("D's m(int,int)");
    }
}
```

① D's m(int,int)；子类D中重写（覆盖）了 参数为两个int类型的m方法，此时调用的是子类重写的方法

② C' m(double,double)；显然

③ C' m(double,double)；这里1.0f会自动转型，float转为double

④ D's m(int,int)；子类引用指向自身，在自身类中找方法，找不到再到父类中去找

⑤ C' m(double,double)；double不能自动转型为float，到D中去找方法

⑥ D's m(float,float)；显然

父类引用指向子类对象这种形式，有以下几种情况：

- 如果子类重写了父类的某个方法，那么此时调用的是子类重写的方法
- 如果子类没有重写父类的某个方法，那么此时调用的是父类中的方法
- 如果子类新增了父类中不存在的方法，那么这个父类引用是不能调用这个仅在子类中存在的方法中，因为子类对象自动向上转型为了父类对象
- 如果子类与父类有同名的成员变量和静态变量，那么由于子类自动向上转型为父类对象，那么输出的必然是父类的成员变量和静态变量
- 子类引用指向自身，子类可以继承父类的成员变量和静态变量，同时可以覆盖父类的成员变量和静态变量

阅读程序写出结果

```
class A {
    public static void m1() {
        System.out.println("A's m1");
    }
    public void m2() {
        System.out.println("A's m2");
    }
}

class B extends A {
    public static void m1() {
        System.out.println("B's m1");
    }
    public void m2() {
        System.out.println("B's m2");
    }
}


class C extends B {
    public static void m1() {
        System.out.println("C's m1");
    }
    public void m2() {
        System.out.println("C's m2");
    }
}
class Test_Hide_Override {
    public static void main(String... args) {
        A o = new C();
        o.m1();                     //①
        o.m2();                     //②
```

```
        ((B) o).m1();               //③
        ((A) (B) o).m1();               //④
        ((A) (B) o).m2();               //⑤
    }
}
```

① A's m1

② C's m2

③ B's m1

④ A's m1

⑤ C's m2

**静态方法不能被override。 而且调用只和引用类形有关， 也就是说你用哪个类的引用（无论heap上存的真实类型是什么），你就是在调用哪个类的static方法，所以多态对静态方法是不适用的。**