java面向对象考题和答案

1.下面代码的输出结果是:

```
public class Main {
    public static void main(String[] args) {
        int n = 100;
        int m = 200;
        System.out.println(f(n,m));
        System.out.println(n);
    }
    public static int f(int m, int n) {
         n = m+n;
         return n;
    }
}
```

A.300
  300

B.100
  100

C.300
  100

D.100
  300

2.程序执行的结果是：

```
public class Test {
 public static void main(String[] args) {
   int x = 6;
   Test p = new Test();
   p.doStuff(x);
   System.out.println(" main:  x = " + x);
 }
 void doStuff(int x) {
   System.out.println(" doStuff: x =" + x++);
 }
}
```

A.doStuff: x =7
 main: x = 7

B.doStuff: x =7
 main: x = 6

C.doStuff: x =6
 main: x = 7

D.doStuff: x =6
 main: x = 6

3.以下关于 JVM 的内存结构描述正确的选项是：

A.类的各种信息在方法区中保存

B.堆用于存放程序运行过程当中所有的局部变量

C.栈用于存储使用 new 关键字创立的对象

D.数组属于 JVM 的内存结构

4.以下代码的输出结果是：

```
public class Test {
 public void print(char c) {
  System.out.println("c");
 }

 public void print(int i) {
  System.out.println("i");
 }

 public void print(double d) {
  System.out.println("d");
 }
 public void print(String s) {
  System.out.println("s");
 }
    public static void main(String[] args) {
```

```
  Test test=new Test();
  test.print('5');
 }
}
```

A.c

5.程序执行的结果是：

```
public class Test {
    String name="Tom";
    public Test(String name){
        name=name;
    }
    public static void main(String [] args){
        Test t = new Test("Jack");
        System.out.println(t.name);
    }
}
```

C.Jack　　D." "

6.关于构造方法，以下说法错误的选项是:

A. 构造方法不可以进行方法重写

B. 构造方法用来实例化一个新的对象

C. 构造方法具有和类名相同的名称

D. 构造方法不返回任何数据类型

7. 关于 Java 中继承的特点，以下说法正确的选项是:

A. 使类的定义复杂化

B. Java 只支持单继承，不可多继承，但可以通过实现接口来到达多继承的目的

C. 子类继承父类的所有成员变量和方法，包括父类的构造方法

D. 不可以多层继承，即一个类不可以继承另一个类的子类

8. 以下代码运行的结果是：

```
class Foo {
 public int a;
 public Foo() {
  a = 3;
 }
 public void addFive() {
  a += 5;
 }
```

```
}
class Bar extends Foo {
 public int a;
 public Bar() {
  a = 8;
 }
 public void addFive() {
  this.a += 5;
 }
}
public class TestFoo {
 public static void main(String[] args) {
  Foo foo = new Bar();
  foo.addFive();
  System.out.println("Value: " + foo.a);
 }
}
```

A. Value: 3

B. Value: 8

C. Value: 13

D. Value: 18

9.以下代码编译和运行的结果是：

```
class Person {
 String name = "No name";

 public Person(String nm) {
  name = nm;
 }
}
class Employee extends Person {
 String empID = "0000";

 public Employee(String id) {
  empID = id;
 }
}
public class EmployeeTest {
 public static void main(String[] args) {
  Employee e = new Employee("4321");
  System.out.println(e.empID);
 }
}
```

A.输出：0000

B.输出：4321

C.代码 public Employee(String id) {行，出现编译错误

D.抛出运行时异常

10. 以下代码的运行结果是：

```
public class Animal {
 public String noise() {
  return "peep";
 }
 public static void main(String[] args) {
  Animal animal = new Dog();
  Cat cat = (Cat)animal;
  System.out.println(cat.noise());
 }
}
class Dog extends Animal {
 public String noise() {
  return "bark";
 }
}
class Cat extends Animal {
```

```
 public String noise() {
  return "meow";
 }
}
```

A.peep

D.抛出运行时异常

11.以下代码编译和运行的结果是：

```
public class A {
 public void start() {
  System.out.println("TestA");
 }
}
public class B extends A {
 public void start() {
  System.out.println("TestB");
 }
 public static void main(String[] args) {
  ( (A) new B( ) ).start();
 }
}
```

A.输出：TestA

B.输出：TestB

C.输出：TestA TestB

D.编译错误

12.请看以下代码：

```
class One {
 void foo() {
 }
}
class Two extends One {
 // insert method here
}
```

以下选项中的代码，放置在<插入代码>处无编译错误的选项是：

A.int foo() { /* more code here */ }

B. protected void foo() { /* more code here */ }

C. public void foo() { /* more code here */ }

D. private void foo() { /* more code here */ }

13.以下选项中，不属于 Java 的访问修饰符的是：

A.private

B.protected

C.friendly

D.public

14. 以下代码的输出结果是：

```
class Foo {
 private int x;
 public Foo(int x) {
  this.x = x;
 }
 public void setX(int x) {
  this.x = x;
 }
 public int getX() {
  return x;
 }
}
public class Gamma {
 static Foo fooBar(Foo foo) {
  foo = new Foo(100);
```

```
  return foo;
 }
 public static void main(String[] args) {
  Foo foo = new Foo(300);
  System.out.print(foo.getX() + "-");
  Foo fooFoo = fooBar(foo);
  System.out.print(foo.getX() + "-");
  System.out.print(fooFoo.getX() + "-");
  foo = fooBar(fooFoo);
  System.out.print(foo.getX() + "-");
  System.out.print(fooFoo.getX());
 }
}
```

A.300-100-100-100-100

15. 以下代码运行的结果是：

```
public class Base {
 public static final String FOO = "foo";
```

```
16. class Sub extends Base {
 public static final String FOO = "bar";
}
 public static void main(String[] args) {
  Base b = new Base();
  Sub s = new Sub();
  System.out.print(Base.FOO);
  System.out.print(Sub.FOO);
  System.out.print(b.FOO);
  System.out.print(s.FOO);
  System.out.print(((Base) s).FOO);
 }
}
```

A.foofoofoofoofoo

16. 关于以下代码说法正确的选项是：

```
public class ItemTest {
  private final int id;
```

```
  public ItemTest(int id) {
    this.id = id;
  }
  public void updateId(int newId) {
    id = newId;
  }
  public static void main(String[] args) {
    ItemTest fa = new ItemTest(42);
    fa.updateId(69);
    System.out.println(fa.id);
  }
}
```

A.编译错误

B.运行时抛出异常

C.运行后，fa 对象属性 id 的值没有改变，应然是 42

D.运行后，fa 对象属性 id 的值改变成新的值 69

17. 请看以下代码编译和运行的结果是：

```
public class Student {
 private String name="sun";
 public static void main(String[] args) {
```

```
  Student[] students=new Student[2];
  System.out.println(students[0].name);
  System.out.println(students.length);
 }
}
```

A.sun 2

B.null 2

C.null 1

D.运行时抛出NullPointerException异常

18. 以下代码的输出结果是：

```
abstract class Vehicle {
 public int speed() {
  return 0;
 }
}
class Car extends Vehicle {
 public int speed() {
  return 60;
 }
}
```

```
class RaceCar extends Car {
 public int speed() {
  return 150;
 }
}
public class TestCar {
 public static void main(String[] args) {
  RaceCar racer = new RaceCar();
  Car car = new RaceCar();
  Vehicle vehicle = new RaceCar();
  System.out.println(racer.speed() + ", " +
car.speed() + ", "
    + vehicle.speed());
 }
}
```

A. 0, 0,0　　B. 150, 60, 0

C. 150, 150, 150　　D. 抛出运行时异常

19. 请看以下代码：

```
public abstract class Employee {
 protected abstract double getSalesAmount();
 public double getCommision() {
```

```
  return getSalesAmount() * 0.15;estA() {
 }
}
class Sales extends Employee {
 <插入代码>
}
```

在<插入代码>处填入的方法正确的选项是：

B D

A. `double getSalesAmount() { return 1230.45; }`

B. `public double getSalesAmount() { return 1230.45; }`

C. `private double getSalesAmount() { return 1230.45; }`

D. `protected double getSalesAmount() { return 1230.45; }`

20.关于以下代码说法正确的选项是：

```
public interface A {
 public void doSomething(String thing);
}
public class AImpl implements A {
 public void doSomething(String msg) {
 }
```

```
}
public class B {
 public A doit() {
  return null;
 }
 public String execute() {
  return null;
 }
}
public class C extends B {
 public AImpl doit() {
  return null;
 }
 public Object execute() {
  return null;
 }
}
```

A. 所有类和接口都编译成功　　B.类 B 编译失败

C.类 Aimpl 编译失败　　D.类 C 编译失败

21.关于以下代码说法正确的选项是：你妈的，什么吊题

```
interface A {
 public void aMethod();
}
interface B {
 public void bMethod();
}
interface C extends A, B {
 public void cMethod();
}
class D implements B {
 public void bMethod() {}
}
class E extends D implements C {
 public void aMethod() {}
 public void bMethod() {}
 public void cMethod() {}
}
```

A.编译失败

B.如果定义 D e = new E();, 那么 e.bMethod();调用 D 类的 bMethod()方法

C.如果定义 D e = (D)(new E());, 那么 e.bMethod();调用 D 类的 bMethod()方法

D.如果定义 D e = (D)(new E());, 那么 e.bMethod();调用 E 类的 bMethod()方法

22.请看以下代码：

```
public class UserRequest {
 public void request(ServletAction action) {
  action.doService();
 }
 public static void main(String[] args) {
  UserRequest user = new UserRequest();
  user.request(new ServletAction() {
   public void doService() {
    System.out.println("处理请求");
   }
  });
 }
}
```

如果上述代码采用回调模式编写，以下关于 ServletAction 的定义正确的选项是：

A. public static class ServletAction {

```
    public void doService();
}
```

B.
```
public final class ServletAction {
    public void doService();
}
```

C.
```
public class ServletAction {
    public void doService();
}
```

D.
```
public interface ServletAction {
    public void doService();
}
```

23. 以下代码运行的结果是：

```
public class Hello {
 String title;
 int value;
 public Hello() {
  title += "World";
 }
 public Hello(int value) {
  this.value = value;
```

```
  title = "Hello";
 }
 public static void main(String[] args) {
  Hello c = new Hello(5);
  System.out.println(c.title);
 }
}
```

A.Hello

B. Hello World

C. Hello World 5

D.运行后无输出

24. 请看以下代码编译和运行的结果是：

```
interface TestA {
 String toString();
}
public class Test {
 public static void main(String[] args) {
  System.out.println(new TestA() {
   public String toString() {
    return "test";
```

```
    }
  });
 }
}
```

A.输出：test

B.输出：null

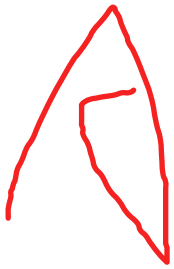
C.代码 System.out.println(new TestA() {行，编译出错

D.代码 public String toString() {行，编译出错

25.请看以下代码：

```
1)public class Outer {
2) void fn(int i) {
3)  class Inner {
4)   void print() {
5)    System.out.println(i);
6)   }
7)  }
8)  Inner in = new Inner();
9)  in.print();
10) }
```

```
11)}
12)class Test {
13)  public static void main(String args[]){
14)   Outer out=new Outer();
15)   out.fn(100);
16)  }
17)}
```



关于上述代码说法正确的选项是:

A.在第 5 行出现编译错误，fn 方法的参数必须用 final 修饰

B.在第 3 行出现编译错误，在方法的内部不能写类

C.在第 8 行出现编译错误，在方法的内部不能实例化 Inner 类的对象

D.运行代码，控制台输出 100

1,正确答案:C

2,正确答案:D

3,正确答案:A

4,正确答案:A

5,正确答案:B

6,正确答案:B

7,正确答案:B

8,正确答案:A

9,正确答案:C

10,正确答案:D

11,正确答案:B

12,正确答案:BC

13,正确答案:C

14,正确答案:B

15,正确答案:D

16,正确答案:A

17,正确答案:D

18,正确答案:C

19,正确答案:BD

20,正确答案:D

21,正确答案:D

22,正确答案:D

23,正确答案:A

24,正确答案:A

25,正确答案:A