# 请大家阅读文档时，在视图里勾选导航窗格，在左边显示章节目录方便浏览。

# 一、填空题

1. 语句Class clz = null；的含义是\_\_\_声明了一个Class类型引用变量clz，该引用初值为空\_\_\_。

2. 给定下列类的定义：

class GeometricObject {}

class PolyGon extends GeometricObject {}

class Rectangle extends PolyGon {}

GeometricObject o = new Rectangle ();

Class clz1 = o. getClass ();

(1) 声明一个指向PolyGon及其子类的类型信息的引用变量clz的语句应该是

\_\_\_\_\_\_\_ Class<? extends Person> clz;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_；

(2) System.out.println(o.getClass().getSimpleName());的输出结果是\_\_\_\_\_\_Rectangle\_\_\_\_\_\_\_\_;

(3) 下列语句中有错误的是\_\_\_\_\_\_\_②\_③\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

Class<PolyGon> clz3 = null;

clz3 = PolyGon.class; ①

clz3 = Rectangle.class; ②

Class<? extends PolyGon> clz4 = null;

clz4 = GeometricObject.class; ③

clz4 = PolyGon.class; ④

clz4 = Rectangle.class; ⑤

错误原因是（按错误题号解释）\_\_\_②：clz3只能指向PolyGon类型的Class，不能指向PolyGon的子类因此报错\_\_\_\_\_\_\_\_③:clz4只能指向PolyGon及其子类而GeometricObject为PolyGon的父类，因此错误\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

3. 下面五条语句中，错误的有\_\_\_（2），（3）\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

（1）ArrayList<String> lists = new ArrayList<String>();

（2）ArrayList<Object> lists = new ArrayList<String>();

（3）ArrayList<String> lists = new ArrayList<Object>();

（4）ArrayList<String> lists = new ArrayList();

（5）ArrayList lists = new ArrayList<String>();

错误原因是\_\_\_\_\_\_虽然String为Object的子类，但ArrayList<String>不是ArrayList<Object>的子类。因此ArrayList<Object>型引用变量只能引用ArrayList<Object>类型的对象,同理ArrayList<Strint>型引用变量只能引用ArrayList<String>类型的对象\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

使用泛型通配符?将错误的语句修改正确的方法是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_（2）ArrayList<? extends Object> lists = new ArrayList<String>();\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_（3）\_ArrayList<? super String> lists = new ArrayList< Object >();\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

4. 下面代码给出了泛型类和非泛型类的定义：

class Holder<T> {

T value;

public Holder (T value) {this.value = value;}

public T getValue () {return value;}

}

class RawHolder {

Object value;

public RawHolder (Object value) {this.value = value;}

public Object getValue () {return value;}

}

基于上面二个类的定义，有下面四段代码：

|  |  |
| --- | --- |
| ①  Holder<String> h1 = new Holder<>("aaa");  String s1 = h1. getValue ();  System.out.println(s1); | ②  RawHolder h1 = new RawHolder("aaa");  String s1 = (String)h1. getValue ();  System.out.println(s1); |
| ③  Holder<String> h1 =  new Holder<> (Integer.valueOf(111));  String s1 = h1. getValue ();  System.out.println(s1); | ④  RawHolder h1 =  new RawHolder (Integer.valueOf(111));  String s1 = (String)h1. getValue ();  System.out.println(s1); |

上面四段代码中编译通过运行不出错的是\_\_\_\_\_\_\_①\_\_②\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_，

上面四段代码中编译通过运行出错的是\_\_\_④\_\_\_，原因是\_\_\_\_RawHolder中的value类型为Object，可以存储类型为Integer类型变量，getValue的返回值为Object，因此通过强制类型转换赋值给s1编译器没有报错。但是运行时由于h1的value值为Integer类型，不能强制转换为String类型，出现运行错误\_\_\_\_\_\_\_\_\_\_\_\_\_，

上面四段代码中编译不通过是\_\_③\_\_\_，原因是\_\_\_\_Holder使用了泛型，Holder<String>类型引用不能指向Holder<Integer>实例，编译出错\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_，

这个例子说明泛型的作用是\_\_\_\_将判断运行时的类型匹配的工作交由编译器辅助判断，减少程序员的工作负担\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

# 二、单项选择题

1. 泛型参数<T>代表的是\_\_\_\_D\_\_\_\_\_\_\_。

A. 任意类型

B. 某类型的子类型

C. 某类型的父类型

D. 固定指代某种类型

2. 泛型通配符<?>代表的是\_\_\_A\_\_\_\_\_\_\_\_。

A. 任意类型

B. 某类型的子类型

C. 某类型的父类型

D. 固定指代某种类型

3. 下面泛型定义中不正确的是\_\_\_\_D\_\_\_\_\_\_\_。

A. class Test1<T> {}

B. interface Test2<T> {}

C. class Test3{<T> void test () {}}

D. class Test4{void <T> test () {}}

4. 泛型通配符<? extends T>代表的是\_\_\_B\_\_\_\_\_\_\_\_。

A. 任意类型

B. 某类型T的子类型

C. 某类型T的父类型

D. 固定指代某种类型

5. 泛型通配符<? super T>代表的是\_\_\_\_\_\_C\_\_\_\_\_。

A. 任意类型

B. 某类型T的子类型

C. 某类型T的父类型

D. 固定指代某种类型

6. 关于下面代码，描述正确的是\_\_\_\_\_C\_\_\_\_\_\_。

List<String> list = new ArrayList<String>();

list.add("test");

list.add("red");

list.add (100);

System.out.println(list. size ());

A. 输出2

B. 输出3

C. 编译错误

D. 运行时报异常

7. 关于下面代码，描述正确的是\_\_\_\_\_\_B\_\_\_\_\_。

List<Integer> ex\_int= new ArrayList<Integer> ();

List<Number> ex\_num = ex\_int;

System.out.println(ex\_num. size ());

A. 0

B. 编译错误

C. 运行时报异常

D. 1

8. 下列语句编译时不出错的是\_\_\_\_D\_\_\_\_\_\_\_。

A. List<?> c1 = new ArrayList<String> (); c1.add (new Object ());

B. List<?> c2 = new ArrayList<String> (); c2.add (new String ("1"));

C. List<?> c3 = new ArrayList<String> (); c3.add ("1");

D. List<?> c4 = new ArrayList<String> (); c4.add(null);

9. 给定下列代码：

class Shape {}

class Circle extends Shape {}

class Triangle extends Shape {}

public class Test2\_9 {

public static void main (String [] args) {

List<? extends Shape> list1 = new ArrayList< Triangle> ();

List<? extends Shape> list2 = new ArrayList<Circle> ();

System.out.println(list1 instanceof List< Triangle>); ①

System.out.println(list2 instanceof List); ②

System.out.println(list1.getClass() == list2.getClass()); ③

}

}

则关于语句①②③说法正确的是：\_\_\_\_\_D\_\_\_\_\_\_。

A. ①②③输出结果为true、false、false

B. ①②③输出结果为true、true、true

C. ①编译出错，②③输出结果为false、false

D. ①编译出错，②③输出结果为true、true

# 三、多项选择题（一个或多个正确选项）

1. 对于泛型类class A<T> { … }，T在A类里可以用作不同的地方，在A类类体内，下面语句正确的有\_\_\_A\_\_B\_\_\_\_D\_\_G\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

A. T x;

B. T m1() {return null;}

C. static T y;

D. void m2(T i) {}

E. static T s1() {return null;}

F. static void s2(T i) {}

G. static <T1> void s3(T1 i, T1 j){}

2. 下列语句编译时不出错的是\_\_\_\_\_A\_E\_\_G\_\_H\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

A. List<? super Integer> x1 = new ArrayList<Number> ();

B. List<? super Number> x2 = new ArrayList<Integer> ();

C. List<? super Number> x3 = new ArrayList<Short> ();

D. List<? super Integer> x4 = new ArrayList<Short> ();

E. List<? extends Number> x5 = new ArrayList<Integer> ();  
F. List<? extends Number> x6 = new ArrayList<Object> ();  
G. List<Number> x7 = new ArrayList<> ();  
H. List<? extends Comparable<Double>> x8 = new ArrayList<Double> ();

I. List<? extends Number> x9 = new ArrayList<int> ();

3. 下面泛型类是List<?>的子类的是\_\_\_\_\_\_\_\_ABC\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

A. List<String>

B. List<Object>

C. List<Integer>

D. List<float>

4. 泛型参数应该写在的位置是\_\_\_\_\_\_\_\_\_B\_\_\_\_\_\_D\_\_\_\_\_\_\_\_\_\_。

A. 类名前

B. 类名后

C. 方法名前

D. 方法返回值类型前

5. 关于java泛型，下面描述正确的是\_\_\_\_\_\_\_\_A\_B\_C D\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

A. 泛型的类型参数只能是类类型（包括自定义类），不能是基本类型

B. 泛型的类型参数可以有多个

C. 不能对泛型的具体实例类型使用instanceof操作，如 o instanceof ArrayList<String>，否则编译时会出错。

D. 不能创建一个泛型的具体实例类型的数组，如 new ArrayList<String>[10]，否则编译时会出错。

6. 给定下列类和泛型方法的定义：

class A {}

class B extends A {}

class C extends B {}

class D extends C {}

public class Test2\_9{

public static <T> void m (List<? super T> list1, List<? extends T> list2) {}

}

则下面6段代码编译出错的是\_\_\_\_\_C\_\_E\_\_F\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

A.

List<B> l1 = new ArrayList<> ();

List<B> l2 = new ArrayList<> ();

Test2\_9.m (l1, l2);

B.

List<B> l3 = new ArrayList<> ();

List<D> l4 = new ArrayList<> ();

Test2\_9.m (l3, l4);

C.

List<B> l5 = new ArrayList<> ();

List<A> l6 = new ArrayList<> ();

Test2\_9.m (l5, l6);

D.

List<C> l7 = new ArrayList<> ();

List<D> l8 = new ArrayList<> ();

Test2\_9.m (l7, l8);

E.

List<C> l7 = new ArrayList<> ();

List<D> l8 = new ArrayList<> ();

Test2\_9. <B>m (l7, l8);

F.

List<D> l9 = new ArrayList<> ();

List<C> l10 = new ArrayList<> ();

Test2\_9.m (l9, l10);

# 四、问答题

阅读下列程序，并填写表格

import java.util.\*;

class A {}

class B extends A {}

class Test {

public static void m1(List<? extends A> list) {}

public static void m2(List<A> list) {}

public static void m3(List<? super A> list) { }

public static void main (String [] args) {

List<A> listA = new ArrayList<A> ();

List<B> listB = new ArrayList<B> ();

List<Object> listO = new ArrayList<Object> ();

// insert code here

}

}

|  |  |
| --- | --- |
| 在上面代码插入点插入的代码 | 结果（从下面结果选项中选择） |
| m1(listA); | C |
| m2(listA); | C |
| m3(listA); | C |
| m1(listB); | C |
| m2(listB); | A |
| m3(listB); | A |
| m1(listO); | A |
| m2(listO); | A |
| m3(listO); | C |
| 结果选项 | |
| A. 编译出错 | |
| B. 编译正确，运行出错 | |
| C.编译正确，运行正确 | |