What will be printed when the following code is compiled and run?

```
public class LoadTest{

    public static void main(String[] args) throws Exception {
        LoadTest t = new LoadTest();
        int i = t.getLoad();
        double d = t.getLoad();
        System.out.println( i + d );
    }

    public int getLoad() {
        return 1;
    }

    public double getLoad(){
        return 3.0;
    }

}
```

No puedes tener más de un método en una clase con la misma firma. La firma del método incluye el nombre del método y la lista de argumentos, pero no incluye el tipo de retorno. Por lo tanto, los dos métodos getLoad() tienen la misma firma y no se compilarán. Esto muestra que la sobrecarga de métodos no se puede realizar en función de los tipos de retorno

//

Consider the following code:

```
public class TestClass{
   public void method(Object o){
      System.out.println("Object Version");
   }
   public void method(java.io.FileNotFoundException s){
      System.out.println("java.io.FileNotFoundException Version");
   }
   public void method(java.io.IOException s){
```

```
    System.out.println("IOException Version");
  }
  public static void main(String args[]){
    TestClass tc = new TestClass();
    tc.method(null);
  }
}
```

What would be the output when the above program is compiled and run? (Assume that FileNotFoundException is a subclass of IOException, which in turn is a subclass of Exception)

It will print `java.io.FileNotFoundException Version`

La razón es bastante simple: se llama al método más específico dependiendo del argumento. Aquí, se puede pasar nulo a los 3 métodos, pero la clase FileNotFoundException es la subclase de IOException, que a su vez es la subclase de Object. Entonces, la clase FileNotFoundException es la clase más específica. Entonces, este método se llama. Si hubiera habido dos métodos más específicos, ni siquiera se compilaría ya que el compilador no podrá determinar a qué método llamar.

//

```
//How can you fix the following code to make it compile:
//import java.io.*;
/*class Great {
   public void doStuff() throws FileNotFoundException{
   }
}
```

```
class Amazing extends Great {
  public void doStuff() throws IOException, IllegalArgumentException{
  }
}

public class TestClass {
    public static void main(String[] args) throws IOException{
        Great g = new Amazing();
        g.doStuff();
    }
}*/
```

//Assume that changes suggested in a option are to be applied independent of other options.

Change `doStuff` in `Amazing` to throw only `IllegalArgumentException`.

Change `doStuff` in `Great` to throw only `IOException` instead of `FileNotFoundException`.

//

What will the following program print when compiled and run?

```
/*class Game{
  public void play() throws Exception{
    System.out.println("Playing...");
  }
}
```

```
public class Soccer extends Game{
   public void play(){
      System.out.println("Playing Soccer...");
   }
   public static void main(String[] args){
      Game g = new Soccer();
      g.play();
   }
*/}
```

No complila por la excepción del método padre

//

Given: //in file Movable.java package p1; public interface Movable {   int location = 0;   void move(int by);   public void moveBack(int by); }   //in file Donkey.java package p2; import p1.Movable; public class Donkey implements Movable{   int location = 200;   public void move(int by) {   location = location+by;   } public void moveBack(int by) {   location = location-by;   } }   //in file TestClass.java package px; import p1.Movable; import p2.Donkey; public class TestClass {   public static void main(String[] args) {   Movable m = new Donkey();   m.move(10);   m.moveBack(20); System.out.println(m.location);   } }

Imprime 0 porque toma la variable de la interfaz que vale 0

//

//What will be the result of compiling and running the following code?

```
/*class Base{
   public short getValue(){ return 1; } //1
}
```

```
class Base2 extends Base{
   public byte getValue(){ return 2; } //2 diferente tipo
}
public class TestClass{
   public static void main(String[] args){
      Base b = new Base2();
      System.out.println(b.getValue()); //3
   }
}*/
```

Compile time error at //2

Porque el método se esta override pero base2 tiene un tipo de reotno direfetente
//

jueves

What will the following code print when compiled and run?

```
 import java.util.*;
public class ClassnameTest {
  public static void main(String[] args) {
 List<String> list = new ArrayList<>();
 StringBuilder sb = new StringBuilder("mrx");
  String s = sb.toString();
 list.add(s);
System.out.println(s.getClass());
 System.out.println(list.getClass());
}
 }
```

class java.lang.String
class java.util.List

class java.lang.String
class java.util.Collection

class java.lang.String
class java.util.ArrayList

class java.lang.Object
class java.util.ArrayList

class java.lang.Object
class java.util.List

//

//Which statements concerning the following code are true?

```
class A{
  public A() {} // A1
  public A(String s) {  this();  System.out.println("A :"+s);  }  // A2
}

class B extends A{
  public int B(String s) {  System.out.println("B :"+s);  return 0; } // B1
}
class C extends B{
    private C(){ super(); } // C1
    public C(String s){  this();  System.out.println("C :"+s);  } // C2
    public C(int i){} // C3
}
```

Respuestas:

At least one of the constructors of each class is called as a result of constructing an object of class C.

Constructor at //A2 will never be called in creation of an object of class C.

Class C can be instantiated only in two ways by users of class C.

//B1 will never be called in creation of objects of class A, B, or C.

None of the constructors of class A will be called in creation of an object of class C.

None of the constructors of class B will be called in creation of an object of class C.

//

Exceptions

What will the following code print when compiled and run?

```
 abstract class Calculator{
abstract void calculate();
public static void main(String[] args){
  System.out.println("calculating");
  Calculator x = null;
x.calculate();
}
 }
```

It will print `calculating` and then throw `NullPointerException`.

//

```
//What will be the output when the following program is run ?
 package exceptions;
 public class TestClass {
    public static void main(String[] args) {
       try {
          doTest();
       } catch (MyException me) {
          System.out.println(me);
       }
    }

    static void doTest() throws MyException {
       int[] array = new int[10];
       array[10] = 1000;
       doAnotherTest();
    }

    static void doAnotherTest() throws MyException {
       throw new MyException("Exception from doAnotherTest");
    }
 }
 class MyException extends Exception {
    public MyException(String msg) {
       super(msg);
    }
 }
//(Assume that there is no error in the line numbers given in the options.)
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 10
    at exceptions.TestClass.doTest(TestClass.java:14)
    at exceptions.TestClass.main(TestClass.java:5)
```

//

//Identify the exceptions that will be received when the following code snippets are //executed.

```
1. int factorial(int n){
  if(n==1) return 1;
 else return n*factorial(n-1);
}
```

Assume that it is called with a very large integer

.

```
 2. void printMe(Object[] oa){
   for(int i=0; i<=oa.length; i++)
   System.out.println(oa[i]);
  }
```

Assume that it is called as such: printMe(null);

```
3. Object m1(){
   return new Object();
  }
 void m2(){
  String s = (String) m1();
  }
```

//Assume that method m2 is invoked.

ClassCastException
ArrayIndexOutOfBoundsException
StackOverflowError

ClassCastException
ArrayIndexOutOfBoundsException
SecurityException

No Exception Will Be Thrown
SecurityException
Will Not Compile

StackOverflowError
NullPointerException
No Exception Will Be Thrown

StackOverflowError
ArrayIndexOutOfBoundsException
ClassCastException

StackOverflowError
NullPointerException
NullPointerException

SecurityException
NullPointerException
No Exception Will Be Thrown

StackOverflowError
NullPointerException
ClassCastException

//

Consider the following hierarchy of Exception classes :

java.lang.RuntimeException

+---- IndexOutOfBoundsException

+----ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException

Which of the following statements are correct for a method that can throw ArrayIndexOutOfBounds as well as StringIndexOutOfBounds Exceptions but does not have try catch blocks to catch the same?

Respuestas

The method calling this method will either have to catch these 2 exceptions or declare them in its `throws` clause.

It is ok if it declares just `throws ArrayIndexOutOfBoundsException`

It must declare `throws ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException`

It is ok if it declares just `throws IndexOutOfBoundsException`

It does not need to declare any throws clause.

//

What will the following class print ?

```
class Test{
   public static void main(String[] args){
      int[][] a = { { 00, 01 }, { 10, 11 } };
      int i = 99;
      try {
         a[val()][i = 1]++;
      } catch (Exception e) {
         System.out.println( i+", "+a[1][1]);
      }
   }
   static int val() throws Exception {
     throw new Exception("unimplemented");
   }
}
```

Respuestas

99 , 11

1 , 11

1 and an unknown value.

99 and an unknown value.

It will throw an exception at Run time.

//

What will be the output when the following program is run?

```
package exceptions;
public class TestClass{
    public static void main(String[] args) {
        try{
            hello();
        }
        catch(MyException me){
            System.out.println(me);
        }
    }

    static void hello() throws MyException{
        int[] dear = new int[7];
        dear[0] = 747;
        foo(); se maneja desde el main y corta el programa
    }

    static void foo() throws MyException{
        throw new MyException("Exception from foo");
    }
```

```
}

class MyException extends Exception {
    public MyException(String msg){
        super(msg);
    }
}
```

(Assume that line numbers printed in the messages given below are correct.)

Respuesta

```
exceptions.MyException: Exception from foo
```

//

//What will be the result of attempting to compile and run the following program?

```
/*class TestClass{
   public static void main(String args[]){
      int i = 0;
      loop :        // 1
      {
         System.out.println("Loop Lable line");
         try{
            for (  ;  true ;  i++ ){
```

```
            if( i >5) break loop;       // 2
          }
        }
        catch(Exception e){
          System.out.println("Exception in loop.");
        }
        finally{
          System.out.println("In Finally");      // 3
        }
      }
    }
}*/
```

Respuestas

No compilation error and line 3 will be executed.

//

What will the following code print when compiled and run? (Assume that MySpecialException is an unchecked exception.)

```
1. public class ExceptionTest {
2.    public static void main(String[] args) {
3.       try {
4.          doSomething();
5.       } catch (MySpecialException e) {
6.          System.out.println(e);
```

```
7.       }
8.    }
9.
10.    static void doSomething() {
11.        int[] array = new int[4];
12.        array[4] = 4;
13.        doSomethingElse();
14.    }
15.
16.    static void doSomethingElse() {
17.        throw new MySpecialException("Sorry, can't do something else");
18.    }
}
```

Respuesta

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 4
    at ExceptionTest.doSomething(ExceptionTest.java:12)
    at ExceptionTest.main(ExceptionTest.java:4)
```

//

What will the following code print?

```
public class Test{
 public int luckyNumber(int seed){
  if(seed > 10) return seed%10;
 int x = 0;
  try{
  if(seed%2 == 0) throw new Exception("No Even no.");
   else return x;
         }
catch(Exception e){
```

```
            return 3;
   }
finally{
   return 7;
 }
 }
 public static void main(String args[]){
            int amount = 100, seed = 6;
switch( new Test().luckyNumber(6) ){
  case 3: amount = amount * 2;
 case 7: amount = amount * 2;
  case 6: amount = amount + amount;
  default :
  }
  System.out.println(amount);        }
 }
```

Respuesta

400

//

```
//What is the result of compiling and running this code ?
/*class MyException extends Throwable {
}
class MyException1 extends MyException {
}
class MyException2 extends MyException {
}
class MyException3 extends MyException2 {
}
public class ExceptionTest {
   void myMethod() throws MyException {
      throw new MyException3();
```

```
    }

    public static void main(String[] args) {
        ExceptionTest et = new ExceptionTest();
        try {
            et.myMethod();
        } catch (MyException me) {
            System.out.println("MyException thrown");
        } catch (MyException3 me3) { primero es el hijo
            System.out.println("MyException3 thrown");
        } finally {
            System.out.println(" Done");
        }
    }
}*/
```

Respuesta

Para la excepción de debe ir de lo general a la particular en este caso es primero la excepción hija

//

```
//What will the following code print when run?
public class Test {
    static String s = "";

    public static void m0(int a, int b) {
        s += a;
        m2();
        m1(b);
    }

    public static void m1(int i) {
        s += i;
    }

    public static void m2() {
        throw new NullPointerException("aa");
    }

    public static void m() {
        m0(1, 2);
        m1(3);
    }
```

```
   public static void main(String args[]) {
      try {
         m();
      } catch (Exception e) {
      }
      System.out.println(s);
   }
}
```

Imprime 1

//

What will the following code snippet print:

```
 Float f = null;
 try{
f = Float.valueOf("12.3");
    String s = f.toString();
    int i = Integer.parseInt(s);  falla aquí
    System.out.println(""+i);
 }
catch(Exception e){
 System.out.println("trouble : "+f);
}
```

Respuesta

trouble : 12.3