UNIQ1

Consider the following code:

```
public class TestClass {
public static void doStuff() throws Exception{
System.out.println("Doing stuff...");
if(Math.random()>0.4){
throw new Exception("Too high!");        }
System.out.println("Done stuff.");     }
public static void main(String[] args) throws Exception {
 doStuff();
System.out.println("Over");
  } }
```

Which of the following are possible outputs when the above program is compiled and run? (Assume that Math.random() returns a double between 0.0 and 1.0 not including 1.0. Further assume that there is no mistake in the line numbers printed in the output shown in the options.)

✗ You answered incorrectly You had to select 2 options

```
Doing stuff...
Exception in thread "main" java.lang.Exception: Too high!
        at TestClass.doStuff(TestClass.java:29)
        at TestClass.main(TestClass.java:41)
```

```
Doing stuff...
Exception in thread "main" java.lang.Exception: Too high!
        at TestClass.doStuff(TestClass.java:29)
        at TestClass.main(TestClass.java:41)
Over
```

If doStuff() throws an exception, the code after the call to doStuff() in main will not be executed and therefore, "Over" will not be printed.

```
Doing stuff...
Done stuff.
Over
```

```
Doing stuff...
Exception in thread "main" java.lang.Exception: Too high!
        at TestClass.doStuff(TestClass.java:29)
        at TestClass.main(TestClass.java:41)
```

Done stuff.

Once an exception is thrown in a method, the code after that exception will not be executed. Therefore, if doStuff() throws an exception, "Done stuff." will not be printed.

There are only two possibilities:

1. If `Math.random()` generates a number more than 0.4, the if part will throw an exception. In this case, the remain code of `doStuff` will not be called and `main()` will receive an exception due to the call to `doStuff`. Since `doStuff()` is not within a try/catch block, the exception will propagate up and the remaining code in `main()` will not be executed either.

Since the exception is not caught anywhere in the code, it will finally reach the JVM's thread that has called the main method. This thread catches the exception and prints out the stack trace.

2. If `Math.random()` generates a number not more than 0.4, if part will not be executed and "`Done stuff.`" will be printed. After the call returns in `main()`, "`Over`" will be printed as well.

//////

Given that the bit pattern for the amount $20,000 is 100111000100000, which of the following is/are valid declarations of an int variable that contains this value?

You had to select 1 option

int b = (binary)100111000100000;

(binary) is invalid because binary is not a valid keyword.

int b = 01001110_00100000;

This is a valid piece of code but the value is not correct. Since it does not start with 0b or 0B, it will NOT be interpreted as a binary number. In fact, since it starts with 0, it will be interpreted as an octal number.

int b = 0b01001110_00100000;

If you want to write a value in binary, it must be prefixed with 0b or 0B.

int b = b1001110_00100000;

///////////////

Which line contains a valid constructor in the following class definition?

```
public class TestClass{
  int i, j;
  public TestClass getInstance() {  return new TestClass();    }  //1
```

```
   public void TestClass(int x, int y) {   i = x;   j = y;   }     //2
   public TestClass TestClass() {   return new TestClass();   }    //3
   public ~TestClass() {     }                    //4
}
```

You had to select 1 option

Line 1

This cannot be a constructor because even the name of the method (getInstance) is not same as the class name!

Line 2

Constructors cannot return anything. Not even void.

Line 3

Constructors cannot return anything. Not even void.

Line 4

This could have been a destructor in C++ world. And there nothing like this in java. Java has a finalize() method, which is similar to a destructor but does not work exactly as a destructor.

None of the above.

///////////////////////

Which of the following lambda expressions can be used to invoke a method that accepts a java.util.function.Predicate as an argument?
You had to select 2 options

x -> System.out.println(x)

The body part of the lambda expression that is meant to capture the Predicate interface must return a boolean (because the only abstract method in this interface returns boolean) but here, it returns void.

x -> System.out.println(x);

1. The body part must return a boolean but this returns void. 2. This is syntactically invalid as well because of the semi-colon.

x -> x == null

() -> true

The implementation of Predicate interface must have a method that takes exactly one parameter. Here, the parameter list is empty.

x->true

////////////////

Consider the following class :

```
public class Parser{
  public static void main( String[] args){
     try{
        int i = 0;
        i =  Integer.parseInt( args[0] );
     }
     catch(NumberFormatException e){
       System.out.println("Problem in " + i );
     }
  }
}
```

What will happen if it is run with the following command line:

java Parser one

You had to select 1 option

It will print Problem in 0

It will throw an exception and end without printing anything.

It will not even compile.

Because 'i' is defined in try block and so it is not visible in the catch block.

It will not print anything if the argument is '1' instead of 'one'.

None of the above.

//////////////////

What will the following code print when run?

import java.util.function.Predicate; class Employee{

```
 int age;   //1 }
 public class TestClass{
 public static boolean validateEmployee(Employee e, Predicate<Employee> p){
return p.test(e);    }
public static void main(String[] args) {
 Employee e = new Employee(); //2       System.out.println(validateEmployee(e, e-
>e.age<10000)); //3    } }
```

You had to select 1 option

It will fail to compile at line marked //1

No problem here. age will automatically be initialized to 0 because it is an instance field of the class.

It will fail to compile at line marked //2

No problem here. Employee class will get a default no-args constructor because it doesn't define any constructor explicitly. Therefore, new Employee() is valid.

It will fail to compile at line marked //3

Remember that the parameter list part of a lambda expression declares new variables that are used in the body part of that lambda expression. However, a lambda expression does not create a new scope for variables. Therefore, you cannot reuse the local variable names that have already been used in the enclosing method to declare the variables in you lambda expression. It would be like declaring the same variable twice. Here, the main method has already declared a variable named e. Therefore, the parameter list part of the lambda expression must not declare another variable with the same name. You need to use another name. For example, if you change //3 to the following, it will work.   System.out.println(validateEmployee(e, x->x.age<10000)); It would print true.

It will compile fine and print true when run.

It will compile fine and print false when run.
//////////////////////

Which of the following can be used as a constructor for the class given below?

```
public class TestClass{
 // lots of irrelevant code } (...
```

in the options means irrelevant code that is not shown here.)

You had to select 2 options

public void TestClass() {...}
There should be no return type. Not even void.

public TestClass() {...}

public static TestClass() {...}
Constructors cannot be static.

public final TestClass() {...}
Constructors cannot be final.

public TestClass(int x) { ...}

You can use only one of public protected and private.
Unlike methods, a constructor cannot be abstract, static, final, native, or synchronized. A constructor is not inherited, so there is no need to declare it final and an abstract constructor could never be implemented. A constructor is always invoked with respect to an object, so it makes no sense for a constructor to be static. There is no practical need for a constructor to be synchronized, because it would lock the object under construction, which is normally not made available to other threads until all constructors for the object have completed their work. The lack of native constructors is an arbitrary language design choice that makes it easy for an implementation of the Java Virtual Machine to verify that superclass constructors are always properly invoked during object creation.

//////////////////
What will the following program print?

```
public class TestClass{
  static String str;
  public static void main(String[] args){
     System.out.println(str);
```

}}

You had to select 1 option

It will not compile.

It will compile but throw an exception at runtime.

It will print 'null' (without quotes).

It will print nothing.

None of the above.

All member fields (static and non-static) are initialized to their default values. Objects are initialized to null (String is also an object), numeric types to 0 (or 0.0) and boolean to false.

//////////////////

Given the following LOCs:

int rate = 10;

XXX amount = 1 - rate/100*1 - rate/100;

What can XXX be?

You had to select 1 option

only int or long

only long or double

only double

double or float

long or double but not int or float.

int, long, float or double

Note that none of the terms in the expression `1 - rate/100*1 - rate/100;` is double or float. They are all ints. So the result of the expression will be an int.

Since an int can be assigned to a variable of type int, long, float or double, amount can be int, long, float or double.

//////////////////////

Identify the correct statements about ArrayList?

You had to select 3 options

Standard JDK provides no subclasses of ArrayList.
It does. Direct Known Subclasses: AttributeList, RoleList, RoleUnresolvedList

An ArrayList cannot store primitives.
This is true because only objects can be stored in it. Note that the following is valid though: List l = new ArrayList(); l.add(1); Although 1 is a primitive, it is valid because it will be wrapped into an Integer object before being passed to the list.

It allows constant time access to all its elements.
This is true because it implements java.util.RandomAccess interface, which is a marker interface that signifies that you can directly access any element of this collection. This also implies that it takes the same amount of time to access any element. (Compare that with a LinkedList, which takes more time to access the last element than the first element.)

ArrayList cannot resize dynamically if you add more number of elements than its capacity.
It does resize dynamically. Compare that to an array, which cannot be resized once created. Although there is a constructor in ArrayList that allows you to limit the maximum size of the ArrayList. But it is not really a limitation of the ArrayList itself but a feature for the user.

An ArrayList is backed by an array.
This is true. The elements are actually stored in an array and that is why is it called an ArrayList. (The expression "backed by an array" means that the implementation of ArrayList actually uses an array to store elements.)

ArrayList is a subclass of AbstractList.

```
java.lang.Object
-  java.util.AbstractCollection<E>
   -    java.util.AbstractList<E>
     -      java.util.ArrayList<E>
```

All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:
AttributeList, RoleList, RoleUnresolvedList
//////////////////////

Consider the following code:

```
class Test{
public static void main(String[] args){
for (int i = 0; i < args.length; i++)
 System.out.print(i == 0 ? args[i] : " " + args[i]);   } }
```

What will be the output when it is run using the following command:

```
java Test good bye friend!
```

You had to select 1 option

[✓ good bye friend!]

good good good

goodgoodgood

good bye

None of the above.

The arguments passed on the command line can be accessed using the args array. The first argument (i.e. good) is stored in args[0], second argument (i.e. bye) is stored in args[1] and so on.
Here, we are passing 3 arguments. Therefore, args.length is 3 and the for loop will run 3 times. For the first iteration, i is 0 and so the first operand of the ternary operator (?) will be returned, which is args[i]. For the next two iterations, " "+args[i] will be returned. Hence, the program will print three strings: "good", " bye", and " friend!" on the same line.

Note that unlike in C++, program name is not the first parameter in the argument list. Java does not need to know the program name because the .class file name and the java class name are always same (for a public class). So the java code always knows the program name it is running in. So there is no need to pass the program name as the first parameter of the argument list. In C/C++, the binary file name may be anything so the code does not know what binary file it is going to end up in. That's why the program name is also sent (automatically) in parameter list.

///////////////

Given:

```
public class TableTest {
static String[][] table;
public static void main(String[] args) {
String[] x = { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun" };
 String[] y1 = { "1", "2", "3", "4", "5" };
String[] y2 = { "a", "b", "c" };
table = new String[3][];
table[0] = x;
table[1] = y1;
 table[2] = y2;
//INSERT CODE HERE
} }
```

What can be inserted in the above code to make it print Sun5c?

You had to select 1 option

```
for(String[] row : table){
System.out.print(row[row.length]); }
```

Remember that since indexing starts with 0, length is always 1 greater than the last index. Therefore, row[row.length] will throw ArrayIndexOutOfBoundsException.

```
int i = 0;
for(String[] col : table){
 i++;
if(i==col.length){
System.out.print(table[col.length][i]);     } }
```

```
for(String[] row : table){
System.out.print(row[row.length-1]); }
```

```
for(int i=0; i<table.length-1; i++){
int j = table[i].length-1;
 System.out.print(table[i][j]); }
```

This is almost correct. Since the for condition is i < table.length-1 It will only print Sun5 and leave out the c.

///////////

A java source file contains the following code:

```
interface I {
 int getI(int a, int b);
}
interface J{
  int getJ(int a, int b, int c);
}
abstract class MyIJ implements J , I { }
class MyI{
  int getI(int x, int y){ return x+y; }
}
interface K extends J{
  int getJ(int a, int b, int c, int d);
}
```

Identify the correct statements:

You had to select 1 option

It will fail to compile because of MyIJ

MyIJ declares that it implements interfaces I and J, but does not implement the methods declared in these interfaces. However, since MyIJ has been declared as abstract, it is valid.

It will fail to compile because of MyIJ and K

It will fail to compile because of K

K is a valid interface because an interface is permitted to extend another interface.

It will fail to compile because of MyI and K

Both are valid.

It will fail to compile because of MyIJ, K, and MyI

It will compile without any error.

///////////////
Which of the following is correct about Java byte code?

You had to select 1 option

It can run on any OS and chip architecture.

It can run on any OS and chip architecture if there is a JRE available for that OS and chip architecture.

It can run only any OS and chip architecture if that platform has a JRE built for it and the Java code was compiled ON that platform.

It can run only any OS and chip architecture if that platform has a JRE built for it and the Java code was compiled FOR that platform

Java byte code is basically just a set of instructions that are intepreted by a virtual machine and is independent of the actual machine and OS i.e. the platform. JRE (Java Runtime Environment) is the virtual machine that interprets the given byte code and converts it into the acutal platform understandable instructions. Therefore, all you need to run the byte code is the virtual machine (JRE) for that specific platform on which you want to run it.
Since the byte code itself is platform independent, you can compile your java code on any platform because no matter where you compile your code, the same byte code will be produced. Therefore, you don't need a java compiler for a particular platform. You just need the JRE for that platform. Oracle provides JRE for several platforms inluding Windows and Unix.
/////////////

Which line(s) of code in the following program will cause a compilation error?

```
public class TestClass{
  static int value = 0; //1
  public static void main(String args[]) //2
  {
    int 2ndArgument = Integer.parseInt(args[2]); //3
    if( true == 2 > 10 ) //4
    {
      value = -10;
    }
    else{
      value =  2ndArgument;
    }
    for( ; value>0; value--) System.out.println("A"); //5
  }
}
```

✕ You answered incorrectly You had to select 1 option

1

2

3

2ndArgument is not a valid identifier name because an identifier must not start with a digit (although it can contain a digit.) An identifier may start with and contain the underscore character _.

4

== has less precedence than > so true == 2 > 10 is same as true == (2 > 10)

5

//////////////////////////

Which of the following are valid ways to create a LocalDateTime?

You had to **select** 1 option

java.time.LocalDate.parse("2015-01-02");

To create an instance of LocalDateTime, you need to use the methods in LocalDateTime class. Methods in LocalDate class create LocalDate instances. Similarly, methods in LocalTime class create LocalTime instances.

java.time.LocalDateTime.parse("2015-01-02");

LocalDateTime requires date as well as time. Here, you just have a date in the input so it will throw a java.time.format.DateTimeParseException.

java.time.LocalDateTime.parse("2015-01-02T17:13:50"); would be valid.

java.time.LocalDateTime.of(2015, 10, 1, 10, 10);

```
java.time.LocalDateTime.of(2015, "January", 1, 10, 10);
```

All parameters should be ints. For the month argument, you can either pass the numbers 1 to 12 (and not 0 to 11) or use constants such as java.time.Month.JANUARY.

/////////////////////

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

```
class SwitchTest{
public static void main(String args[]){
 for ( int i = 0 ; i < 3 ; i++){
boolean flag  = false;
switch (i){
flag  = true;         }
if ( flag )  System.out.println( i );       }    } }
```

You had to select 1 option

It will print 0, 1 and 2.

It will not print anything.

Compilation error.

It will say 'case', 'default' or '}' expected at compile time.

Runtime error.

None of the above.

You cannot have unlabeled block of code inside a `switch` block. Any code block must succeed a case label (or default label). Since there is no `case` statement in this switch block, there is no way the line `flag = true;` can be reached! Therefore, it will not compile.

/////////////////

Which of the statements regarding the following code are correct?

```
public class TestClass{
static int a;
int b;
public TestClass(){
int c;
c = a;
a++;
b += c;    }
public static void main(String args[]) {
new TestClass();   } }
```

You had to select 1 option

The code will fail to compile because the constructor is trying to access static members.
A constructor (or any other method) can access static members.

The code will fail to compile because the constructor is trying to use static member variable a before it has been initialized.
static fields are always initialized automatically if you do not initialize them explicitly. So are instance fields.

The code will fail to compile because the constructor is trying to use member variable b before it has been initialized.

The code will fail to compile because the constructor is trying to use local variable c before it has been initialized.
c is getting initialized at line 2: c = a;

The code will compile and run without any problem.

All the instance or static variables are given a default values if not explicitly initialized.
All numeric variable are given a value of zero or equivalent to zero (i.e. 0.0 for double or float).
booleans are initialized to false and objects references are initialized to null.
//////////////////////
What will the following lines of code print?

```
System.out.println(1 + 5 < 3 + 7);
System.out.println( (2 + 2) >= 2 + 3);
```

You had to select 1 option

They will not compile.

1false10
false

true
false

false
false

Comparison operators have lower precedence than mathematical operators. Therefore, 1 + 5 < 3 + 7 is evaluated as (1 + 5) < (3 + 7) i.e. 6<10, which prints true.
Similarly, (2 + 2) >= 2 + 3 is evaluated as (2 + 2) >= (2 + 3) i.e. 4>=5, which prints false.

If you have an expression, 2 + (2 >= 2) + 3, it would be tempting to answer 2true3, but actually, it would not compile because it would resolve to 2 + true + 3 and + operator is not overloaded for anything except String. Here, neither of the operands of + operator is a String.
///////////////

Consider the following class...

```
class TestClass{
   void probe(Object x) { System.out.println("In Object"); } //3

   void probe(Number x) { System.out.println("In Number"); } //2

   void probe(Integer x) { System.out.println("In Integer"); } //2

   void probe(Long x) { System.out.println("In Long"); } //4

   public static void main(String[] args){
      double a = 10;
      new TestClass().probe(a);
   }
}
```

What will be printed?

You had to select 1 option

In Number

In Object

In Long

In Integer

It will not compile.

Here, we have four overloaded probe methods but there is no probe method that takes a `double` parameter. However, a `double` will be boxed into a `Double` and class `Double` extends `Number`. Therefore, a `Double` can be passed to the method that takes `Number`. A Double can also be passed to a method that takes `Object`, but `Number` is more specific than Object therefore `probe(Number )` will be called.

We advise you to run this program and try out various combinations. The exam has questions on this pattern.

//////////////////////

What will the following program print?

```
public class TestClass{
  static boolean b;
  static int[] ia = new int[1];
  static char ch;
  static boolean[] ba = new boolean[1];
  public static void main(String args[]) throws Exception{
    boolean x = false;
    if( b ){
      x = ( ch == ia[ch]);
    }
    else x = ( ba[ch] = b );
    System.out.println(x+" "+ba[ch]);
  }
}
```

You had to select 1 option

true true

true false

false true

false false

It will not compile.

This question tests your knowledge on the default values of uninitialized primitives and object references. `booleans` are initialized to `false`, numeric types to 0 and object references to `null`. Elements of arrays are initialized to the default values of their types. So, elements of a `boolean` array are initialized to `false`. int, char, float to 0 and Objects to null.

In this case, `b` is `false`. So the else part of `if(b)` is executed.
`ch` is a numeric type so its value is `0`. `ba[0] = false` assigns `false` to `ba[0]` and returns `false` which is assigned to `x`.
Finally, `x` and `ba[0]` are printed as `false false`.
/////////////////
Given:

```
interface I { }
class A implements I{
   public String toString(){ return "in a"; }
}
class B extends A{
   public String toString(){ return "in b"; }
}
public class TestClass {
   public static void main(String[] args) {
      B b = new B();
      A a = b;
      I i = a;

      System.out.println(i);
      System.out.println((B)a);
      System.out.println(b);
   }
}
```

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

You had to select 1 option

in i
in a
in b

I
A
in b

in a
in a
in b

in a
in b
in b

in b
in b
in b

There is only one object created in this code and the class of that object is B. Therefore, B's toString will be called no matter what reference you use. Therefore, it is print "in b" for all the cases.

If you answered this question incorrectly, you need to understand the concept of polymorphism. We suggest you to go through any book to understand it thoroughly because there are several questions in the exam on similar pattern.
In a nutshell, polymorphism means that it is always the class of the object (and not the class of the reference variable that a variable points to) that determines which method will be called at run time. The concept of polymorphism doesn't apply to private methods or static methods because these methods are never inherited.
.////////////////

Which statements about the following code are correct?

```
interface House{
public default String getAddress(){
```

```
return "101 Main Str";   } }
 interface Bungalow extends House{
public default String getAddress(){
return "101 Smart Str";   } }
class MyHouse implements Bungalow, House{  }
public class TestClass {
public static void main(String[] args) {
House ci = new MyHouse();  //1
 System.out.println(ci.getAddress()); //2   } }
```

You had to select 1 option

Code for interface House will cause compilation to fail.

Code for interface Bungalow will cause compilation to fail.

Code for class MyHouse will cause compilation to fail.

Line at //1 will cause compilation to fail.

Line at //2 will cause compilation to fail.

The code will compile successfully.

There is no problem with the code. It is perfectly valid for a subinterface to override a default method of the base interface. A class that implements an interface can also override a default method.
It is valid for `MyHouse` to say that it implements `Bungalow` as well as `House` even though `House` is redundant because `Bungalow` is a `House` anyway.

It will actually print `101 Smart str`.
/////////////////

Which of the following are features of Java?
Some candidates have reported a similar question being asked with a slightly different (and ambiguous) wording -
Which of the following are objected oriented features of Java?

You had to select 2 options

Object is the root class of all the classes.

Objects cannot be reused.

Not sure what "reuse" really means here but our guess is that it is incorrect. One can certainly call methods on the same object again and again. One can also keep a reference to an object as long as needed.

Objects can share behavior with other objects.

Another very vague statement but it is probably true because two objects of two different classes may share some behavior if the two classes inherit from the same super class.

Everything in Java is an Object.

This is not true because Java does have primitives as well (byte, short, char, int, long, float, double, boolean), which are not Objects.

/////////////////////

Which line, if any, will give a compile time error ?

```
void test(byte x){
  switch(x){
    case 'a':   // 1
    case 256:   // 2
    case 0:     // 3
    default :   // 4
    case 80:    // 5
  }
}
```

You had to select 1 option

Line 1 as 'a' is not compatible with byte.

int value of 'a' can easily fit into a byte.

Line 2 as 256 cannot fit into a byte.

No compile time error but a run time error at line 2.

Line 4 as the default label must be the last label in the switch statement.

Any order of case statements is valid.

There is nothing wrong with the code.

Every case constant expression in a switch block must be assignable to the type of switch expression. Meaning :

```
byte by = 10;
switch(by){
     case 300 :  //some code;
     case 56 :   //some code;
}
```

This will not compile as 300 is not assignable to 'by ' which can only hold values from -128 to 127. This gives compile time error as the compiler detects it while compiling. The use of break keyword is not mandatory, and without it the control will simply fall through the labels of the switch statement.

////////////////////////

Consider the following class...

```
class TestClass{
 int i;
public TestClass(int i) { this.i = i;  }
public String toString(){       if(i == 0) return null;
else return ""+i;    }
public static void main(String[ ] args){
TestClass t1 = new TestClass(0);
 TestClass t2 = new TestClass(2);
 System.out.println(t2);
System.out.println(""+t1);    } }
```

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

You had to select 1 option

It will throw NullPointerException when run.

It will not compile.

It will print 2 and then will throw NullPointerException

It will print 2 and null.

None of the above.

The method `print()/println()` of `PrintStream` takes an `Object` and prints out a String that is returned by calling `String.valueOf(object)`, which in turn calls `toString()` on that object. Note that as `toString()` is defined in Object class, all objects in java have this method. So it prints 2 first.

The second object's `toString()` returns `null`, so it prints "`null`". There is no `NullPointerException` because no method is called on `null`.

Now, the other feature of `print/println` methods is that if they get `null` as input parameter, they print "`null`". They do not try to call `toString()` on `null`.

So, if you have, `Object o = null; System.out.println(o);` will print `null` and will not throw a `NullPointerException`.

///////////////////////

Which of the following correctly defines a method named stringProcessor that can be called by other programmers as follows: stringProcessor(str1) or stringProcessor(str1, str2) or stringProcessor(str1, str2, str3), where str1, str2, and str3 are references to Strings.

You had to select 1 option

public void stringProcessor(...String){ }

public void stringProcessor(String... strs){ }

public void stringProcessor(String[] strs){ }

public void stringProcessor(String a, String b, String c){ }

Three separate methods need to be written.

To allow a method to take variable arguments of a type, you must use the `...` syntax: `methodName( <type>... variableName);`

Remember that there can be only one vararg argument in a method. Further, the vararg argument must be the last argument.

So this is invalid: `stringProcessor( String... variableName, int age);`

but this is valid: `stringProcessor(int age, String... variableName);`

Though not important for this exam, it is good to know that within the method, the vararg argument is treated like an array:

```
public void stringProcessor(String... names){
   for (String n : names) {
       System.out.println("Hello " + n);
   }
}
```

/////////

Consider :

```
 class A {  public void perform_work(){}  }
 class B extends A {  public void perform_work(){}  }
 class C extends B {  public void perform_work(){}  }
```

How can you let perform_work() method of A to be called from an instance method in C?

! Left Unanswered You had to select 1 option

((A) this).perform_work( );

super.perform_work( );

super.super.perform_work( );

this.super.perform_work( );

It is not possible.

The method in C needs to call a method in a superclass two levels up. But `super` is a keyword and not an attribute so `super.super.perform_work( )` strategy will not work. There is no way to go more than one level up for methods.
Remember that this problem doesn't occur for instance variables because variable are never overridden. They are hidden (by instance variables of a subclass) or shadowed (by local variables or method parameters of a method). So to access any of the super class's variable, you must unhide it using a cast. For example, `((A) c).data;` This will give you the data variable defined in A even if it is hidden in B as well as in C.

/////////////////
Consider the following class:

```
public class PortConnector{
public PortConnector(int port) throws IOException{
 ...lot of valid code.    }
...other valid code. }
```

You want to write another class CleanConnector that extends from PortConnector. Which of the following statements should hold true for CleanConnector class?

You had to select 1 option

It is not possible to define CleanConnector that does not throw IOException at instantiation.

It is possible. You can also throw a superclass of IOException from the CleanConnector's constructor. For example, the following is valid: class CleanConnector extends PortConnector { public CleanConnector(int port) throws Exception { super(port); } }

PortConnector class itself is not valid because you cannot throw any exception from a constructor.

A constructor is free to throw any exception.

CleanConnector's constructor cannot throw any exception other than IOException.

It can throw any exception but it must also throw IOException (or its super class). So the following is valid: class CleanConnector extends PortConnector { public CleanConnector(int port) throws IOException, FileNotFoundException, SomeOtherCheckedException { super(port); } }

CleanConnector's constructor cannot throw any exception other than subclass of IOException.

As described above, it can throw any exception but it must throw IOException (or its superclass) as well.

CleanConnector's constructor cannot throw any exception other than superclass of IOException.

As described above, it can throw any exception but it must throw IOException (or its superclass) as well.

None of these.

Observe that the rule for overriding a method is opposite to the rule for constructors. An overriding method cannot throw a superclass exception, while a constructor of a subclass cannot throw subclass exception (Assuming that the same exception or its super class is not present in the subclass constructor's throws clause). For example: class A{ public A() throws IOException{ } void m() throws IOException{ } } class B extends A{ //IOException is valid here, but FileNotFoundException is invalid public B() throws IOException{ } //FileNotFoundException is valid here, but Exception is invalid void m() throws FileNotFoundException{ } } (Note: FileNotFoundException is a subclass of IOException, which is a subclass of Exception) If the subclass constructor's throws clause includes the same exception or its superclass, then it can throw any other exception as well.

As `PortConnector` has only one constructor, there is only one way to instantiate it. Now, to instantiate any subclass of `PortConnector`, the subclass's constructor should call `super(int)`. But that throws `IOException`. And so this exception (or its super class) must be defined in the throws clause of subclass's constructor. Note that you cannot do something like:

```
public CleanConnector(){
   //WRONG : the call to super must be first statement in constructor
   try{ super(); }catch(Exception e){}
}
```

Remember: Constructor must declare all the checked exceptions declared in the base constructor (or the super classes of the checked exceptions). They may add other exceptions as well. This behavior is exactly opposite from that of methods. The overriding method cannot throw any checked exception other than what the overridden method throws. It may throw subclasses of those exceptions as well.

/////////////////////

Consider the following two classes (in the same package but defined in different source files):

```
public class Square {
   double side = 0;
   double area;
   public Square(double length){      this.side = length;   }
   public double getSide() {  return side;   }
   public void setSide(double side) {  this.side = side;   }
   double getArea() {   return area;   }
}
public class TestClass {
   public static void main(String[] args) throws Exception {
      Square sq = new Square(10.0);
      sq.area = sq.getSide()*sq.getSide();
      System.out.println(sq.getArea());
   }}
```

You are assigned the task of refactoring the Square class to make it better in terms of encapsulation. What changes will you make to this class?

You had to select 2 options

Make setSide() method private.

Make getArea() method private.
It should be made public so that other classes can get the area.

Make side and area fields private.
There is no need to keep the area field because that would amount to duplicating the data. If you change side, the value of area will become obsolete.

Make the side field private and remove the area field.

Change getArea method to:
public double getArea(){ return side*side; }

Add a setArea() method.
This is not required because area is calculated using the side. So if you allow other classes to set the area, it could make side and area inconsistent with each other.

There can be multiple ways to accomplish this. The exam asks you questions on the similar pattern.
The key is that your data variable should be private and the functionality that is to be exposed outside should be public. Further, your setter methods should be coded such that they don't leave the data members inconsistent with each other.
.....,,,,,,/////////////////////////////
Consider the following classes :

```
class A{
  public void mA(){ };
}
class B extends A {
  public void mA(){ }
  public void mB() { }
}
class C extends B {
  public void mC(){ }
}
```

and the following declarations:
A x = new B(); B y = new B(); B z = new C();
Which of the following calls are virtual calls?
You had to select 3 options

x.mA();

x.mB();

y.mA();

z.mC();

z.mB();

A virtual call means that the call is bound to a method at run time and not at compile time.

In Java, all non-private and non-final instance method calls are virtual. This is important because, at run time, a reference variable may point to an instance of a subclass of the class of the reference. The compiler doesn't have the knowledge of the class of the actual object being referred to by the reference variable. If the subclass overrides the method, the call becomes polymorphic because now there are two versions of the method that can be invoked (the base class version and the subclass version). Therefore, the compiler is unable to bind the call to the method of a specific class. Only the JVM has the necessary information to bind the call. The JVM knows the class of the actual object and it binds the call to the method of that class. This behavior is called polymorphism.

Thus, in Java, all non-private and non-final instance method calls are potentially polymorphic because there could be multiple versions of the method eligible to be invoked.

In this case, `x.mB()` is invalid call. It will not even compile because the class of x is A, which does not contain method `mB()`. Even though the object referred to by x is of class B which does contain `mB()`. `z.mC()` is invalid for the same reason.

///////////////////

What will the following code print?

```
 int x = 1;
int y = 2;
int z = x++;
 int a = --y;
int b = z--;
b += ++z;
int answ = x>a?y>b?y:b:x>z?x:z;
 System.out.println(answ);
```

You had to select 1 option

0

1

2

-1

-2

3

This is a simple but frustratingly time consuming question. Expect such questions in the exam.
For such questions, it is best to keep track of each variable on the notepad after executing each line of code.

The final values of the variables are as follows -

```
x=2 y=1 z=1 a=1 b=2
```

The expression `x>a?y>b?y:b:x>z?x:z;` should be grouped as -

```
x > a  ? (y>b ? y : b)  :  (x>z ? x : z);
```

It will, therefore, assign 2 to answ.

///////////////////

Given: public class SimpleLoop {
public static void main(String[] args) {
int i=0, j=10;
 while (i<=j) {
i++;   j--;   }   S
ystem.out.println(i+" "+j);
} } What is the result?
You had to select 1 option
6 4

6 5

6 6

5 3

5 4

5 5

In such type of questions, you will need to work out the values of the loop variables for every iteration (unless you can recognize the pattern) on your worksheet.
Beginning i=0, j=10
Iteration 1: i<=j is true, i becomes 1 and j becomes 9
Iteration 2: i<=j is true, i becomes 2 and j becomes 8
Iteration 3: i<=j is true, i becomes 3 and j becomes 7
Iteration 4: i<=j is true, i becomes 4 and j becomes 6
Iteration 5: i<=j is true, i becomes 5 and j becomes 5
Iteration 6: i<=j is true, i becomes 6 and j becomes 4
Iteration 7: i<=j is false so the while loop is not entered.
Print 6 and 4.
////////////////
What will be the output of the following lines ?

```
 System.out.println("" +5 + 6);   //1
System.out.println(5 + "" +6);   // 2
System.out.println(5 + 6 +"");   // 3
System.out.println(5 + 6);       // 4
```

You had to select 1 option

[✓ 56 56 11 11]

11 56 11 11

56 56 56 11

56 56 56 56

56 56 11 56

In line 1, `"" + 5 + 6 => "5"+6 => "56"`
In line 2, `5 + "" +6  => "5"+6 => "56"`
In line 3, `5 + 6 +"" => 11+"" => "11"`
In line 4, `5 + 6 => 11 => "11"`

////////////////////

Given: //Insert code here
public abstract void draw(); }
 //Insert code here
 public void draw(){
System.out.println("in draw..."); } }
Which of the following lines of code can be used to complete the above code?

You had to select 2 options

class Shape {
and
class Circle extends Shape {
Since there is an abstract method in the first class, the class must be declared abstract.

public class Shape {
and
class Circle extends Shape {

abstract Shape {
and
public class Circle extends Shape {
class keyword is missing from the first declaration.

public abstract class Shape {
and
class Circle extends Shape {

public abstract class Shape {
and
class Circle implements Shape {
You can only implement an interface not a class. So Circle implements shape is wrong.

```
public interface Shape {
and
class Circle  implements Shape {
```

By default all the methods of an interface are public and abstract so there is no need to explicitly specify the "abstract" keyword for the draw() method if you make Shape an interface. But it is not wrong to do so.

/////////////////////

What will the following code print?

```
List s1 = new ArrayList( );
s1.add("a");
s1.add("b");
s1.add("c");
s1.add("a");
System.out.println(s1.remove("a")+" "+s1.remove("x"));
```

You had to select 1 option

1 0

2 -1

2 0

1 -1

true false

ArrayList's remove(Object) method returns a boolean. It returns true if the element is found in the list and false otherwise. The JavaDoc API description of this method is important for the exam -

```
public boolean remove(Object o)
```

Removes the first occurrence of the specified element from this list, if it is present (optional operation). If this list does not contain the element, it is unchanged. More formally, removes the element with the lowest index i such that (o==null ? get(i)==null : o.equals(get(i))) (if such an element exists). Returns true if this list contained the specified element (or equivalently, if this list changed as a result of the call).

Observe that it does not remove all occurences of the element. It removes just the first one. In this case, only the first "a" will be removed.

///////////////////////

```
Given:  package strings;
public class StringFromChar {
public static void main(String[] args) {
 String myStr = "good";
char[] myCharArr = {'g', 'o', 'o', 'd' };
 String newStr = "";
for(char ch : myCharArr){
 newStr = newStr + ch;        }
 boolean b1 = newStr == myStr;
 boolean b2 = newStr.equals(myStr);
System.out.println(b1+ " " + b2);            } }
```

What will it print when compiled and run?

You had to select 1 option

true true

true false

false true

false false

In every iteration of the loop, a new String object is created by appending the character to the existing String object. This new String object is assigned back to `newStr`. At the end of the loop, `myStr` is `"good"`, which is why `newStr.equals(myStr)` prints `true`. Since, `newStr` points to a different String object than the one pointed to by `myStr`, `newStr == myStr` evaluates to `false`.

Had `newStr` been defined as `String newStr = null;`, the program would have printed `false` for `newStr == myStr` because both the references are pointing to two different objects, and `false` for `newStr.equals(myStr)` because `newStr` would then contain `"nullgood"`.

/////////////////////////

Given the following code, which statements can be placed at the indicated position without causing compile and run time errors?

```
public class Test{
int i1;
static int i2;
public void method1(){
int i;
  // ... insert statements here    } }
```

You had to select 3 option

i = this.i1;

As i1 is an instance variable, it is accessible through 'this'.

i = this.i2;

Although 'this' is not needed to access i2, it is not an error to do so.

this = new Test( );

Nope, you can't change this.

this.i = 4;

You cannot do this.i as i is a local variable.

this.i1 = i2;

You are just assigning a static field's value to non-static field.

//////////////////

Consider the following code appearing in Eagle.java

```
class Bird {
   private Bird(){     }
}
class Eagle extends Bird {
   public String name;
   public Eagle(String name){
      this.name = name;
   }

   public static void main(String[] args) {
      System.out.println(new Eagle("Bald Eagle").name);
   }
}
```

What can be done to make this code compile?

You had to select 1 option

Nothing, it will compile as it is.

Make Eagle class declaration public:
public class Eagle extends Bird { ... }

Make the Eagle constructor private:
private Eagle(String name){ ... }

Make Bird constructor public:
public Bird() { ... }

Insert super(); as the first line in Eagle constructor:
public Eagle(String name){
super();
this.name = name;   }
If a subclass class constructor doesn't explicitly call the super class constructor, the compiler automatically inserts super(); as the first statement of the subclass constructor. So this option is not needed.

Since the constructor of Bird is private, the subclass cannot access it and therefore, it needs to be made public. protected or default access is also valid.
..///////////////////
What will be the result of attempting to compile and run the following code?

```
class TestClass{
public static void main(String args[] ){
String str1 = "str1";
String str2 = "str2";
System.out.println( str1.concat(str2) );
System.out.println(str1);    } }
```

You had to select 1 option

The code will fail to compile.

The program will print str1 and str1.

The program will print str1 and str1str2

The program will print str1str2 and str1

str1.concat(str2) actually creates a new object that contains "str1str2". So it does not affect the object referenced by str1.

The program will print str1str2 and str1str2.

Note that String objects are immutable. No matter what operation you do, the original object will remain the same and a new object will be returned. Here, the statement str1.concat(str2) creates a new String object which is printed but its reference is lost after the printing.

/////////////////

What will the following program print when compiled and run:

```
public class TestClass {
 public static void main(String[] args) {
 someMethod();     }
static void someMethod(Object parameter)  {
System.out.println("Value is "+parameter);     } }
```

You had to select 1 option

It will not compile.

To call someMethod(Object parameter), you must pass exactly one parameter. So someMethod() is not a valid call to this method and the code will not compile. Note that parameter will not be assigned a null or default value. However, if the method were declared to take variable number of arguments, it would have been valid to call it without any parameters. For example: public static void someMethod(Object... params){   System.out.println(params.length);   } In this case, calling someMethod() without any parameter will print 0. i.e. the length of params array will be 0. params will NOT be null.

Value is null

Value is

It will throw a NullPointerException at run time.

///////////////////////

The following program will print
java.lang.ArithmeticException: / by zero

```
class Test{
  public static void main(String[] args){
    int d = 0;
    try{
      int i = 1 / (d* doIt());
    } catch (Exception e){
      System.out.println(e);
    }
  }
  public static int doIt() throws Exception{
    throw new Exception("Forget It");
  }
}
```

False

It will print `Forget It` because before the division can take place `doIt()` will throw an exception.
///////////////////

What will the following program print when run using the command line:
java TestClass

```
public class TestClass {
public static void methodX() throws Exception {
throw new AssertionError();    }
public static void main(String[] args) {
 try{
methodX();      }
catch(Exception e) {
System.out.println("EXCEPTION");      }    } }
```

You had to select 1 option

It will throw AssertionError out of the main method.

A subclass of Error cannot be caught using a catch block for Exception because java.lang.Error does not extend java.lang.Exception.

It will print EXCEPTION.

The catch block will not be able to catch the Error thrown by methodX().

It will not compile because of the throws clause in methodX().

The throws clause is valid even though unnecessary in this case.

It will end without printing anything because assertions are disabled by default.

It is true that assertions are disabled by default however, methodX is throwing an AssertionError explicitly like any other Throwable. Here, the assertion mechanism is not even used.

//////////////////////

Consider the following two classes (in the same package but defined in different source files):

```
public class Square {
   double side = 0;
   double area;

   public Square(double length){      this.side = length;    }

   public double getSide() {  return side;    }

   public void setSide(double side) {  this.side = side;   }

   double getArea() {   return area;   }
}
public class TestClass {
   public static void main(String[] args) throws Exception {
      Square sq = new Square(10.0);
      sq.area = sq.getSide()*sq.getSide();
      System.out.println(sq.getArea());
   }
}
```

You are assigned the task of refactoring the Square class to make it better in terms of encapsulation. What changes will you make to this class?

You had to select 4 options

Add a calculateArea method:
private void calculateArea(){
this.area = this.side*this.side;

Make side and area fields private.

Change setSide method to:
public void setSide(double d){
this.side = d;
calculateArea(); }

Make the getArea method public.

Add a setArea() method:
public void setArea(double d){ area = d; }
This is not required because area is calculated using the side. So if you allow other classes to set the area, it could make side and area inconsistent with each other.

There can be multiple ways to accomplish this. The exam asks you questions on the similar pattern.
The key is that your data variable should be private and the functionality that is to be exposed outside should be public. Further, your setter methods should be coded such that they don't leave the data members inconsistent with each other.
//////////////////////

Consider the following method -

```
public float parseFloat( String s ){
float f = 0.0f;
try{
 f = Float.valueOf( s ).floatValue();
return f ;    }
catch(NumberFormatException nfe){
f = Float.NaN ;
return f;    }
finally{
f = 10.0f;
 return f;    } }
```

What will it return if the method is called with the input "0.0" ?

You had to select 1 option

It will not compile.

It will return 10.0

It will return Float.Nan

It will return 0.0

None of the above.

finally block will always execute (except when there is a System.exit() in try or catch). And inside the finally block, it is setting f to 10.0. So no matter what the input is, this method will always return 10.0.
////////////////

Identify correct constructs.

You had to select 1 option

```
try {
  for( ;; );
}finally { }
```

A try block must be accompanied by either a catch block or a finally block or both.

```
try {
  File f = new File("c:\\a.txt");
} catch {  f = null; }
```

Invalid syntax for catch. A catch must have a exception: catch(SomeException se){ }

```
int k = 0;
try {
  k = callValidMethod();
}
System.out.println(k);
catch {  k = -1; }
```

There cannot be any thing between a catch or a finally block and the closing brace of the previous try or catch block.

```
try {
  try {
     Socket s = new ServerSocket(3030);
  }catch(Exception e) {
    s = new ServerSocket(4040);
  }
}
```

The first try doesn't have any catch or finally block. Further, the variable s is not in scope in the catch block.

```
try {
       s = new ServerSocket(3030);
}
catch(Exception t){ t.printStackTrace(); }
catch(IOException e) {
    s = new ServerSocket(4040);
}
catch(Throwable t){ t.printStackTrace();  }
```

You can have any number of catch blocks in any order but each catch must be of a different type. Also, a catch for a subclass exception should occur before a catch block for the superclass exception. Here, IOException is placed before Throwable, which is good but Exception is placed before IOException, which is invalid and will not compile.

```
int x = validMethod();
try {
  if(x == 5) throw new IOException();
  else if(x == 6) throw new Exception();
}finally {
 x = 8;
}
catch(Exception e){ x = 9; }
```

finally cannot occur before any catch block.

Note that a try with resources block may or may not to have any catch or finally block at all. However, try with resources is not in scope for this exam.

/////////////

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

```
class Super{
  public int getNumber( int a){
     return 2;
  }
}
public class SubClass extends Super{
  public int getNumber( int a, char ch){
     return 4;
  }
  public static void main(String[] args){
    System.out.println( new SubClass().getNumber(4) );
  }
}
```

What will be printed?

You had to select 1 option

4

2

It will not compile.

It will throw an exception at runtime.

None of the above.

Note that the parameters of SubClass's getNumber are different than Super's getNumber. So it is not overriding it. So the super class's `getNumber()` will be called which returns 2.

/////////////////

What will the following program print?

```
class Test{
    public static void main(String args[]){
        int i=0, j=0;
        X1: for(i = 0; i < 3; i++){
           X2: for(j = 3; j > 0; j--){
              if(i < j) continue X1;
              else break X2;
           }
        }
        System.out.println(i+" "+j);
    }
}
```

You had to select 1 option

0 3

0 2

3 0

3 3

2 2

The statement: `if(i < j) continue X1; else break X2;` only makes sure that the inner loop does not iterate more than once. i.e. for each iteration of i, j only takes the value of 3 and then the j loop terminates, either because of `continue X1;` or because of `break X2;`.
Now, the point to remember here is that when the loop `for(i = 0; i < 3; i++)` ends, the value of i is 3 and not 2.
Similarly, if there were no statement inside inner loop, the value of j after the end of the loop would have been 0 and not 1.

////////////

What will the following code print when compiled and run?

```
public class Test{
 static int a = 0;
 int b = 5;
 public void foo(){
 while(b>0){
b--;
a++;        }    }
public static void main(String[] args) {
 Test p1 = new Test();
 p1.foo();
 Test p2 = new Test();
p2.foo();
System.out.println(p1.a+" "+p2.a);           } }
```

You had to select 1 option

0 10

[✓ 10 10]

10 0

5 5

0 5

5 0

The field `a` is static and there will be only one copy of `a` no matter how many instances of Test you create. Changes made to it by one instance will be reflected in the other instance as well.
But field `b` is an instance field. Each instance of Test will get its on copy of b.

Therefore, when you call p1.foo() and then p2.foo(), the same field `a` is incremented 5 times twice and so it will print `10 10`.

//////////////////

A new Java programmer has written the following method that takes an array of integers and sums up all the integers that are less than 100.

```
 public void processArray(int[] values){
 int sum = 0;
 int i = 0;
try{
while(values[i]<100){
sum = sum +values[i];
i++;           }       }
catch(Exception e){ /* Ignore */ }
 System.out.println("sum = "+sum);     }
```

Which of the following are best practices to improve this code?

You had to select 2 options

Use ArrayIndexOutOfBoundsException for the catch argument.

The intention of the code is to keep iterating the while loop as long as there are elements in the values array. Every iteration increments the index i and once i becomes equal to the length of the values array, values[i] will throw an ArrayIndexOutOfBoundsException, thereby ending the loop. Since the code depends on the raising of an ArrayIndexOutOfBoundsException, that is the exception class that should be used in the catch block. Raising of an ArrayIndexOutOfBoundsException is not really an exceptional situation for this code. Therefore, there is no need to log it. The code does not expect any other class of exception to be raised during the execution and therefore, if any exception other than ArrayIndexOutOfBoundsException is raised, that will truely be an exceptional situation. Such an exception should either not be caught at all (i.e. declared in the throws clause) or if the code catches any such exception, then it should at least be logged.

Use ArrayIndexOutOfBoundsException for the catch argument and add code in the catch block to log or print the exception.

Empty catch blocks are generally a bad practice because at run time, if the exception is thrown, the program will not show any sign of the exception and may produce bad results that will be hard to debug. Therefore, it is a good practice to at least print out the exception if you don't want to do any thing upon encountering an exception. However, in this case, since the code is deliberaly written such a way that an exception will be thrown, the timing and cause of the exception are already known. Therefore, there is no need for logging the exception.

Add code in the catch block to handle the exception.

There are a few questions in the exam that are difficult to interpret. In this case, for example, it is not clear what is meant by handling the exception. The catch block itself is meant to handle the exception. Once you get the exception, you can do what ever is required in the catch block.

Use flow control to terminate the loop.

It is considered a bad practice to use exceptions to control the flow of execution. In this case, values[i] will throw an ArrayIndexOutOfBoundsException once it goes beyond the array length and the programmer is using this fact to control the loop. Instead of doing this, the programmer should use something like: for(int i=0; i<values.length; i++) to control the execution of the loop.

///////////////

Given the following code fragment, which of the following lines would be a part of the output?

```
outer:
  for ( int i = 0 ; i<3 ; i++ ){
    for ( int j = 0 ; j<2 ; j++ ){
      if ( i == j ){
        continue outer;
      }
      System.out.println( "i=" + i + " , j=" + j );
    }
  }
```

You had to select 2 options

i = 1, j = 0

i = 0, j = 1

i = 1, j = 2

i = 2, j = 1

i = 2, j = 2

The given code prints:

```
i=1, j=0
i=2, j=0
i=2, j=1
```

The variable i iterates through the values 0, 1 and 2 in the outer loop, while j varies from 0 to 1 in the inner loop.

If the values of i and j are equal, the continue statement is executed and printing is skipped and next iteration of outer 'for' loop starts.

///////////////

Given the following declaration, select the correct way to get the number of elements in the array, assuming that the array has been initialized.

int[] intArr;

You had to select 1 option

intArr[ ].length( )

intArr.length( )

intArr.length
Each array object has a member variable named public final length of type 'int' that contains the size of the array.

intArr[ ].size( )

intArr.size( )

FYI, All types of arrays are objects. i.e. `intArr instanceof Object` is `true`.
/////////
Which of the following is/are valid instantiations and initializations of a multi dimensional array?
You had to select 2 options

int[][] array2D = new int[][] { { 0, 1, 2, 4} {5, 6}};
It is missing a comma between 4} and {5. It should be: new int[][] { { 0, 1, 2, 4} , {5, 6}};

int[][][] array3D = {{0, 1}, {2, 3}, {4, 5}};
The right side has only two dimensions while the left has three. It should be: int [] [] [] array3D = { { {0, 1}, {2, 3}, {4, 5} } };

int[] array2D[] = new int [2] [2];
array2D[0] [0] = 1;
array2D[0] [1] = 2;
array2D[1] [0] = 3;
Notice that element [1][1] is not given a value explicitly in the code. It is given a default value of 0 automatically.

int[][] array2D = new int[][]{0, 1};
The right side has only one dimension while the left has two. It should be: int [] [] array2D = new int[][]{ {0}, {1}};

```
int[] arr = {1, 2};
int[][] arr2 = {arr, {1, 2}, arr};
 int[][][] arr3 = {arr2};
////////////////
```

Given the following code:

```
public class TestClass {
 public static void main(String[] args) {
//INSERT CODE HERE
System.out.println(x);     } }
```

What can be inserted in the above code so that it will compile and run without any problem?

You had to select 3 options

```
double x = 0xb10_000;
```

0x implies the following digits must be interpreted as Hexadecimal digits and b is a valid Hexadecimal digit.

```
float x = 0b10_000;
```

A number starting with 0b (or 0B) implies that it is written in binary. Since 10000 can fit into a float, an explicit cast is not required. Note that when you specify the bit pattern using binary or hex, an explicit cast is not required even if the number specified using the bit pattern is larger than what a float can hold.

```
float x = 0b20_000;
```

Since it starts with 0b, that means you are writing the number in binary digits (i.e. 0 or 1). But 2 is not a valid binary digit.

```
float x = 0b10_000f;
```

This is invalid because the floating point suffices f, F, d, and D are used only when using decimal system and not while using binary. However, since f is a valid digit in hexadecimal system, a hex number may end with an f although it will not be interpreted as float but as the digit f. Thus, float x = 0x10_000f; and float x = 10_000f; are valid because they are written in hex and decimal respectively but float x = 0b10_000f; is invalid because is written in binary. Note that a floating point number cannot be written in Octal. Therefore, float x = 010_000f; is valid but it is not octal even though it starts with a 0. It is interpreted in decimal.

```
long x = 0b10000L;
```

double d = 0b10_000D;

A floating point number written in binary or hex cannot use any suffix for float. But a floating point number written in decimal can use the floating point suffices f, F, d, and D. Thus, float dx = 0xff; is valid but the f here is not for indicating that it is a float but is interpreted as the hex digit F.

The real exam contains a few questions that test you on how to write numbers in binary. You might want to go through Section 3.10.1 and 3.10.2 of Java Language Specification to understand how this works.

/////////////////////

What will happen when the following program is compiled and run?

```
public class SM{
 public String checkIt(String s){
if(s.length() == 0 || s == null){
return "EMPTY";      }
else return "NOT EMPTY";     }
public static void main(String[] args){
SM a = new SM();
System.out.println(a.checkIt(null));     } }
```

You had to select 1 option

It will print EMPTY.

It will print NOT EMPTY.

It will throw NullPointerException.

Because the first part of the expression (s.length() == 0) is trying to call a method on s, which is null. The check s == null should be done before calling a method on the reference.

It will print EMPTY if || is replaced with |.

In this case, replacing || with | will not make any difference because s.length() will anyway be called before checking whether s is null or not. The right expression would be: if(s == null || s.length() == 0) { ... } In this case, || being a short circuit expression, s.length() == 0 will not be called if s == null returns true. Hence, no NullPointerException will be thrown.

//

What will the following program print when compiled and run?

```
class Data {
   private int x = 0;
   private String y = "Y";
   public Data(int k){
      this.x = k;
   }
   public Data(String k){
      this.y = k;
   }
   public void showMe(){
      System.out.println(x+y);
   }
}
public class TestClass {
   public static void main(String[] args) throws Exception {
      new Data(10).showMe();
      new Data("Z").showMe();
   }
}
```

You had to select 1 option

0Z 10Y

10Y 0Z

You are creating two different Data objects in the code. The first one uses a constructor that takes an integer and the second one uses a constructor that takes a String. Thus, when you call showMe on the first object, it prints 10Y because "Y" is the default value of y as per the given code. When you call showMe on the second object, it prints 0Z because 0 is the default value of x as per the given code.

It will not compile.

It will throw an exception at run time.

Note that + operator is overloaded for String. So if you have a String as any operand for +, a new combined String will be created by concatenating the values of both the operands. Therefore, x+y will result in a String that concatenates integer x and String y.