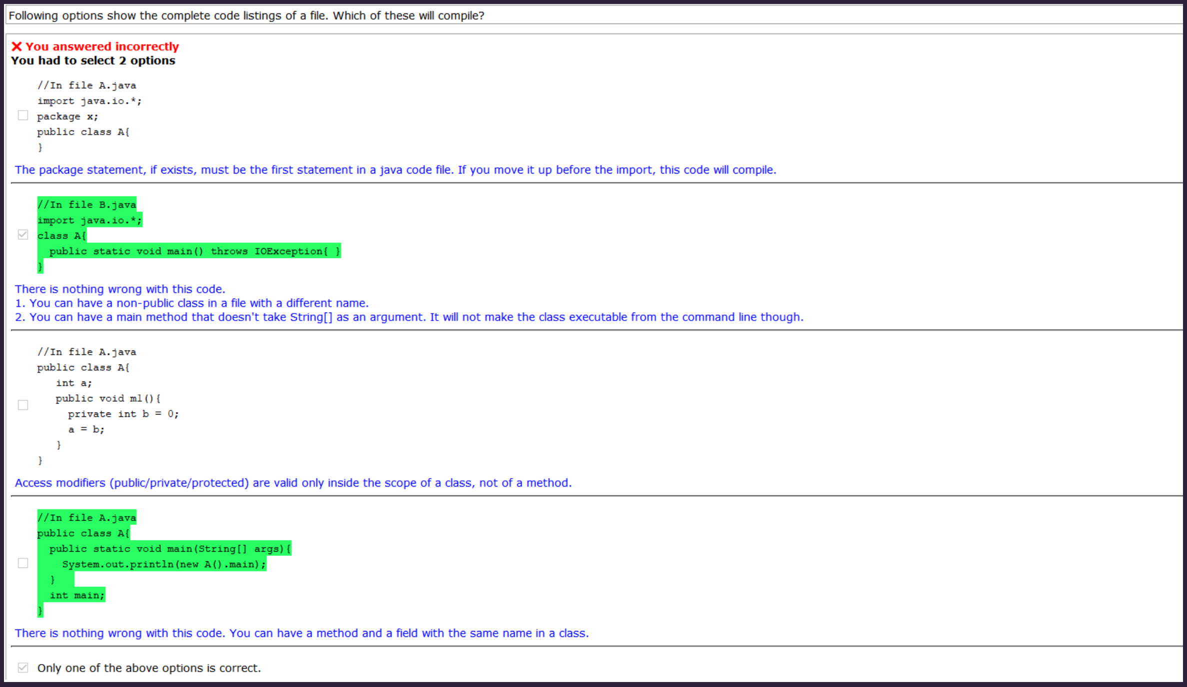
PREGUNTAS ERRÓNEAS PARA REPASAR

01-JAVA BASICS



Following options show the complete code listings of a file. Which of these will compile?

✗ You answered incorrectly
You had to select 2 options

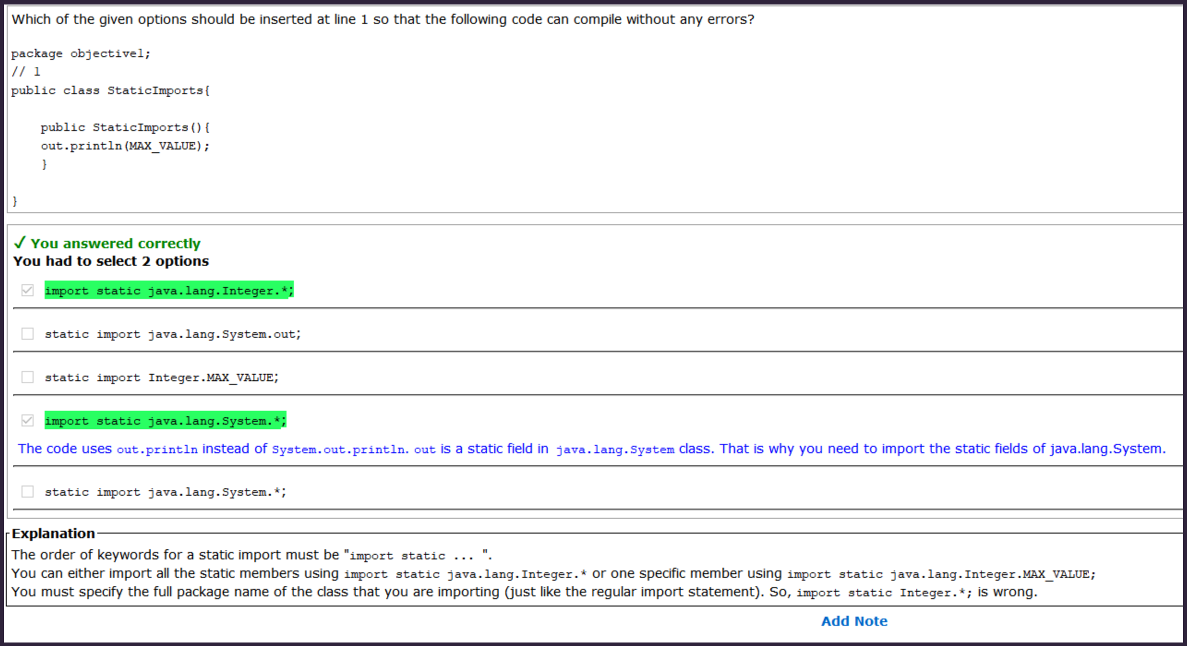
☐
```
//In file A.java
import java.io.*;
package x;
public class A{
}
```
The package statement, if exists, must be the first statement in a java code file. If you move it up before the import, this code will compile.

☑
```
//In file B.java
import java.io.*;
class A{
  public static void main() throws IOException{ }
}
```
There is nothing wrong with this code.
1. You can have a non-public class in a file with a different name.
2. You can have a main method that doesn't take String[] as an argument. It will not make the class executable from the command line though.

☐
```
//In file A.java
public class A{
   int a;
   public void m1(){
     private int b = 0;
     a = b;
   }
}
```
Access modifiers (public/private/protected) are valid only inside the scope of a class, not of a method.

☐
```
//In file A.java
public class A{
  public static void main(String[] args){
    System.out.println(new A().main);
  }
  int main;
}
```
There is nothing wrong with this code. You can have a method and a field with the same name in a class.

☑ Only one of the above options is correct.



Which of the given options should be inserted at line 1 so that the following code can compile without any errors?

```
package objective1;
// 1
public class StaticImports{

    public StaticImports(){
    out.println(MAX_VALUE);
    }

}
```

✓ You answered correctly
You had to select 2 options

☑ `import static java.lang.Integer.*;`

☐ `static import java.lang.System.out;`

☐ `static import Integer.MAX_VALUE;`

☑ `import static java.lang.System.*;`

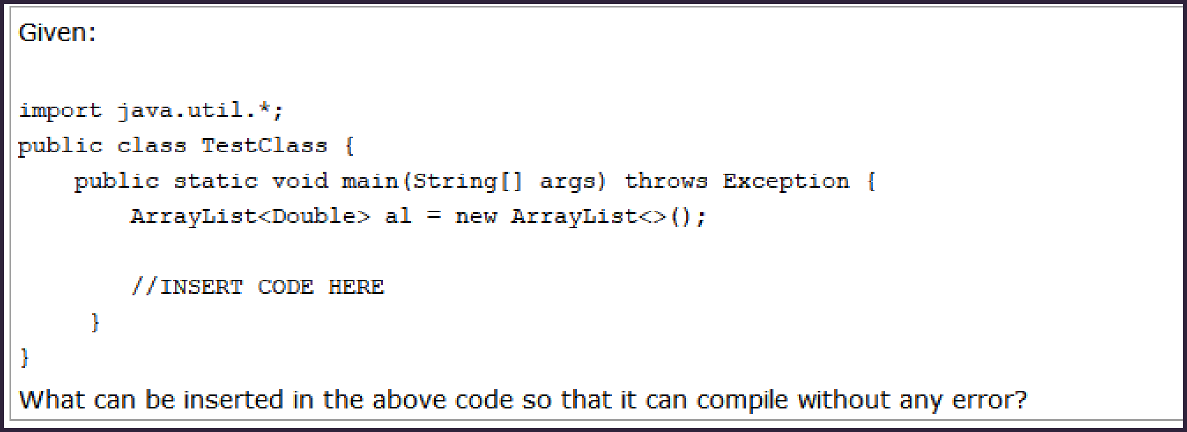
The code uses `out.println` instead of `System.out.println`. `out` is a static field in `java.lang.System` class. That is why you need to import the static fields of java.lang.System.

☐ `static import java.lang.System.*;`

Explanation
The order of keywords for a static import must be "`import static ...` ".
You can either import all the static members using `import static java.lang.Integer.*` or one specific member using `import static java.lang.Integer.MAX_VALUE;`
You must specify the full package name of the class that you are importing (just like the regular import statement). So, `import static Integer.*;` is wrong.

Add Note

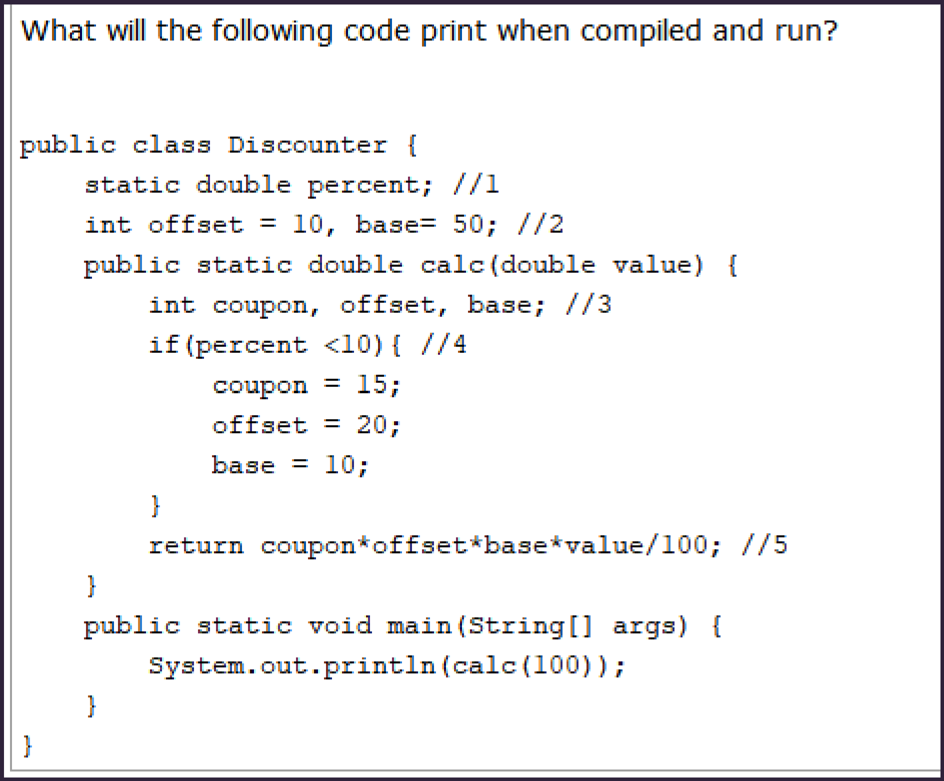
02-WORKING WITH JAVA DATA TYPES


Given:

```
import java.util.*;
public class TestClass {
    public static void main(String[] args) throws Exception {
        ArrayList<Double> al = new ArrayList<>();

        //INSERT CODE HERE
     }
}
```

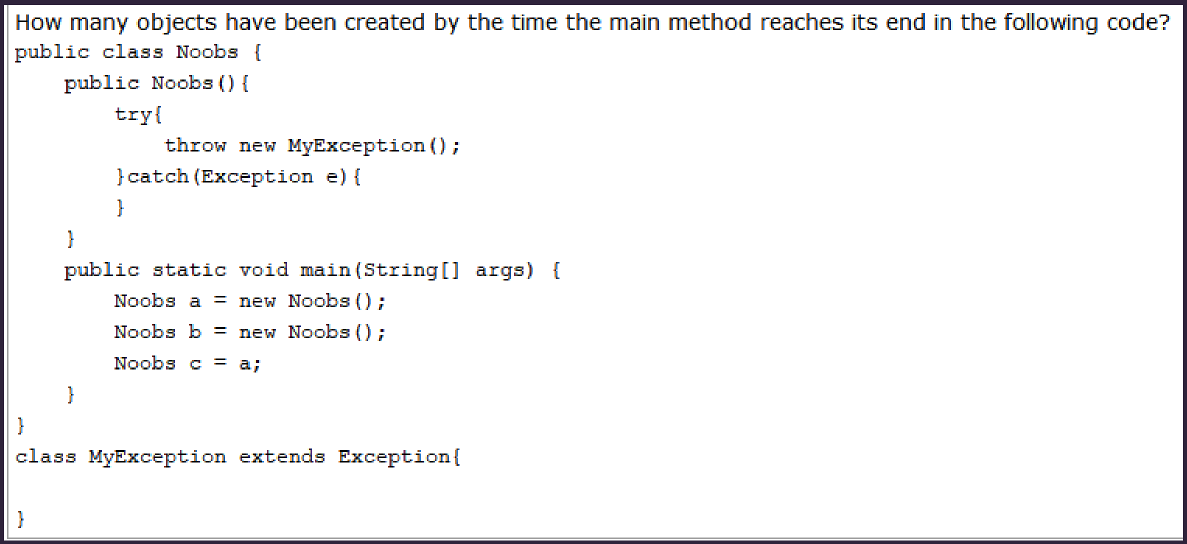
What can be inserted in the above code so that it can compile without any error?

a) al.add(111); No hace cast
b) System.out.println(al.indexOf(1.0));
c) System.out.println(al.contains("string"));
d) Double d = al.get(al.length); NO TIENE ESE ATRIBUTO y al estar vació lanza una excepción.


What will the following code print when compiled and run?

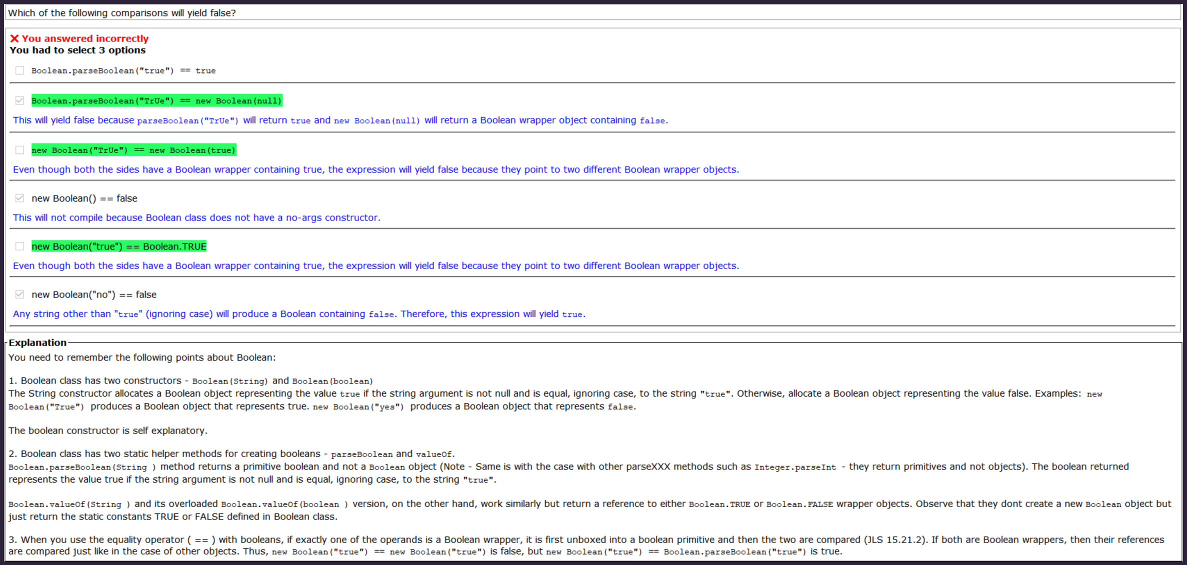
```
public class Discounter {
    static double percent; //1
    int offset = 10, base= 50; //2
    public static double calc(double value) {
        int coupon, offset, base; //3
        if(percent <10){ //4
            coupon = 15;
            offset = 20;
            base = 10;
        }
        return coupon*offset*base*value/100; //5
    }
    public static void main(String[] args) {
        System.out.println(calc(100));
    }
}
```

No compila porque no se inicializa las variables locales del método



How many objects have been created by the time the main method reaches its end in the following code?

```
public class Noobs {
    public Noobs(){
        try{
            throw new MyException();
        }catch(Exception e){
        }
    }
    public static void main(String[] args) {
        Noobs a = new Noobs();
        Noobs b = new Noobs();
        Noobs c = a;
    }
}
class MyException extends Exception{

}
```



Which of the following comparisons will yield false?

✕ You answered incorrectly
You had to select 3 options

[ ] `Boolean.parseBoolean("true") == true`

[✓] `Boolean.parseBoolean("TrUe") == new Boolean(null)`

This will yield false because `parseBoolean("TrUe")` will return `true` and `new Boolean(null)` will return a Boolean wrapper object containing `false`.

[ ] `new Boolean("TrUe") == new Boolean(true)`

Even though both the sides have a Boolean wrapper containing true, the expression will yield false because they point to two different Boolean wrapper objects.

[✓] new Boolean() == false

This will not compile because Boolean class does not have a no-args constructor.

[ ] new Boolean("true") == Boolean.TRUE

Even though both the sides have a Boolean wrapper containing true, the expression will yield false because they point to two different Boolean wrapper objects.

[✓] new Boolean("no") == false

Any string other than "`true`" (ignoring case) will produce a Boolean containing `false`. Therefore, this expression will yield `true`.

Explanation

You need to remember the following points about Boolean:

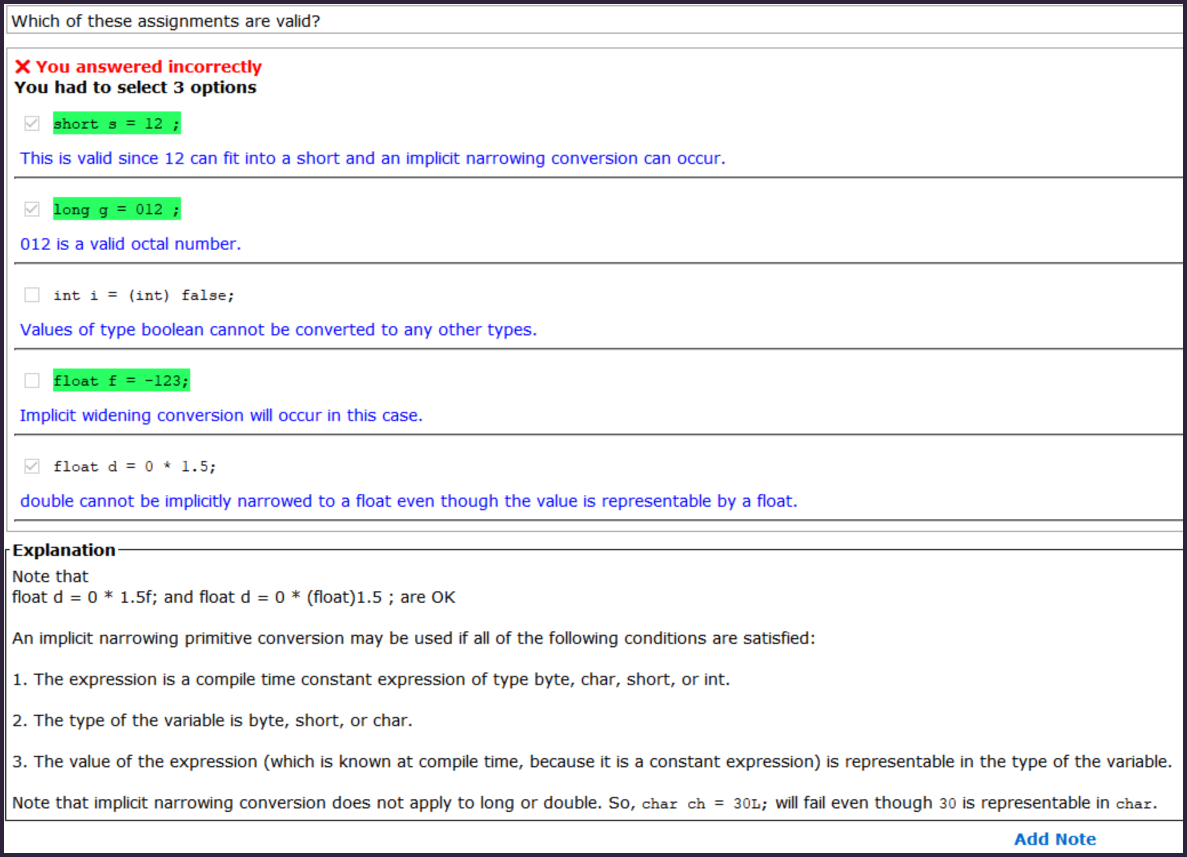
1. Boolean class has two constructors - `Boolean(String)` and `Boolean(boolean)`
The String constructor allocates a Boolean object representing the value `true` if the string argument is not null and is equal, ignoring case, to the string `"true"`. Otherwise, allocate a Boolean object representing the value false. Examples: `new Boolean("True")` produces a Boolean object that represents true. `new Boolean("yes")` produces a Boolean object that represents `false`.

The boolean constructor is self explanatory.

2. Boolean class has two static helper methods for creating booleans - `parseBoolean` and `valueOf`.
`Boolean.parseBoolean(String )` method returns a primitive boolean and not a `Boolean` object (Note - Same is with the case with other parseXXX methods such as `Integer.parseInt` - they return primitives and not objects). The boolean returned represents the value true if the string argument is not null and is equal, ignoring case, to the string `"true"`.

`Boolean.valueOf(String )` and its overloaded `Boolean.valueOf(boolean )` version, on the other hand, work similarly but return a reference to either `Boolean.TRUE` or `Boolean.FALSE` wrapper objects. Observe that they dont create a new `Boolean` object but just return the static constants TRUE or FALSE defined in Boolean class.

3. When you use the equality operator (==) with booleans, if exactly one of the operands is a Boolean wrapper, it is first unboxed into a boolean primitive and then the two are compared (JLS 15.21.2). If both are Boolean wrappers, then their references are compared just like in the case of other objects. Thus, `new Boolean("true") == new Boolean("true")` is false, but `new Boolean("true") == Boolean.parseBoolean("true")` is true.


Which of these assignments are valid?

✗ You answered incorrectly
You had to select 3 options

✓ `short s = 12 ;`

This is valid since 12 can fit into a short and an implicit narrowing conversion can occur.

✓ `long g = 012 ;`

012 is a valid octal number.

☐ `int i = (int) false;`

Values of type boolean cannot be converted to any other types.

☐ `float f = -123;`

Implicit widening conversion will occur in this case.

✓ `float d = 0 * 1.5;`

double cannot be implicitly narrowed to a float even though the value is representable by a float.

Explanation

Note that
float d = 0 * 1.5f; and float d = 0 * (float)1.5 ; are OK

An implicit narrowing primitive conversion may be used if all of the following conditions are satisfied:

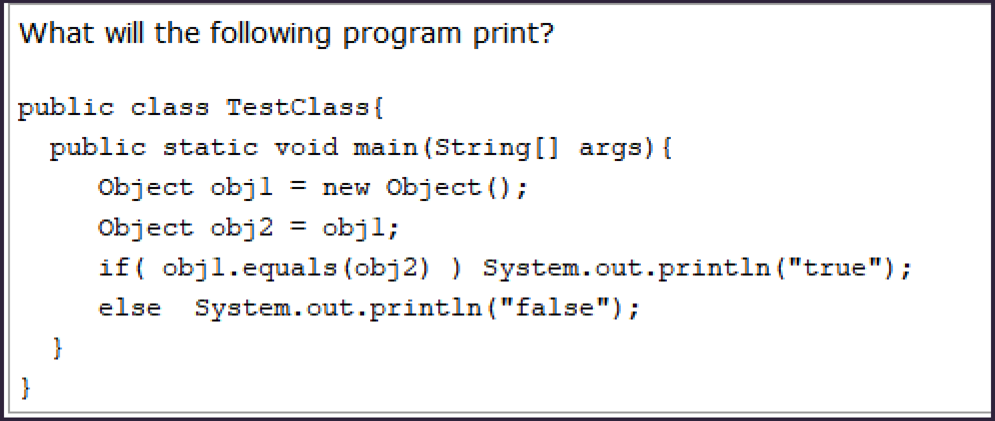
1. The expression is a compile time constant expression of type byte, char, short, or int.

2. The type of the variable is byte, short, or char.

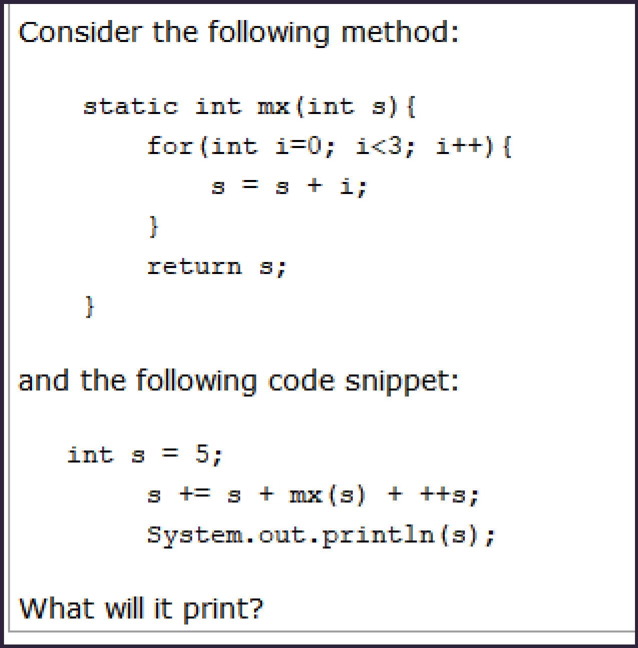
3. The value of the expression (which is known at compile time, because it is a constant expression) is representable in the type of the variable.

Note that implicit narrowing conversion does not apply to long or double. So, `char ch = 30L;` will fail even though `30` is representable in `char`.

03-USING OPERATORS AND DECISION CONSTRUCTS


What will the following program print?

```
public class TestClass{
  public static void main(String[] args){
     Object obj1 = new Object();
     Object obj2 = obj1;
     if( obj1.equals(obj2) ) System.out.println("true");
     else  System.out.println("false");
  }
}
```



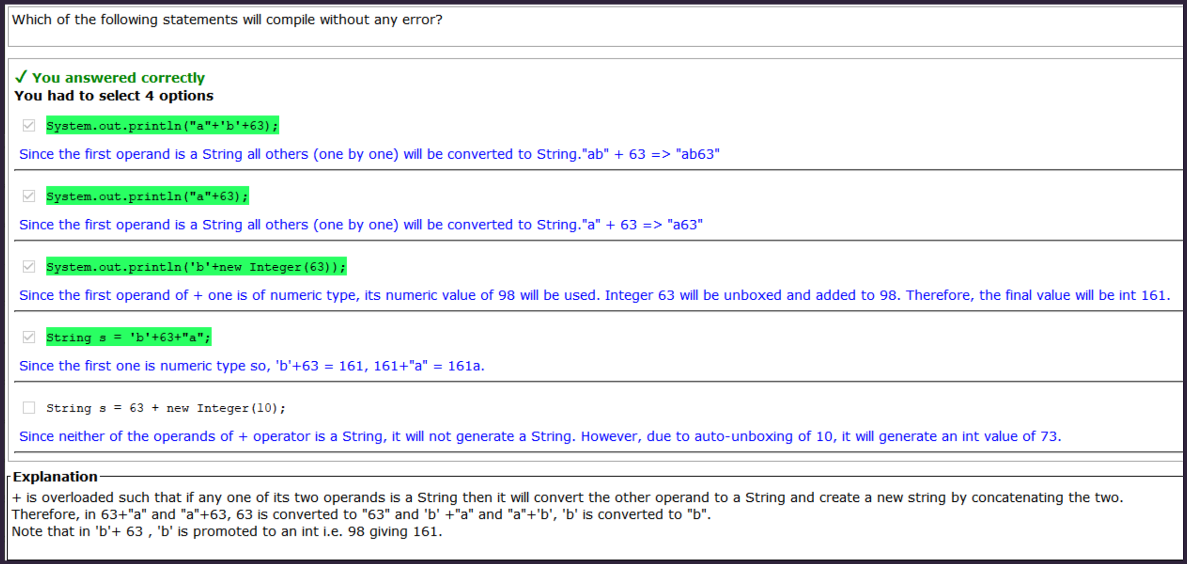
Consider the following method:

```
static int mx(int s){
    for(int i=0; i<3; i++){
        s = s + i;
    }
    return s;
}
```

and the following code snippet:

```
int s = 5;
    s += s + mx(s) + ++s;
    System.out.println(s);
```

What will it print?



Which of the following statements will compile without any error?

✓ You answered correctly
You had to select 4 options

[✓] `System.out.println("a"+'b'+63);`

Since the first operand is a String all others (one by one) will be converted to String."ab" + 63 => "ab63"

[✓] `System.out.println("a"+63);`

Since the first operand is a String all others (one by one) will be converted to String."a" + 63 => "a63"

[✓] `System.out.println('b'+new Integer(63));`

Since the first operand of + one is of numeric type, its numeric value of 98 will be used. Integer 63 will be unboxed and added to 98. Therefore, the final value will be int 161.

[✓] `String s = 'b'+63+"a";`

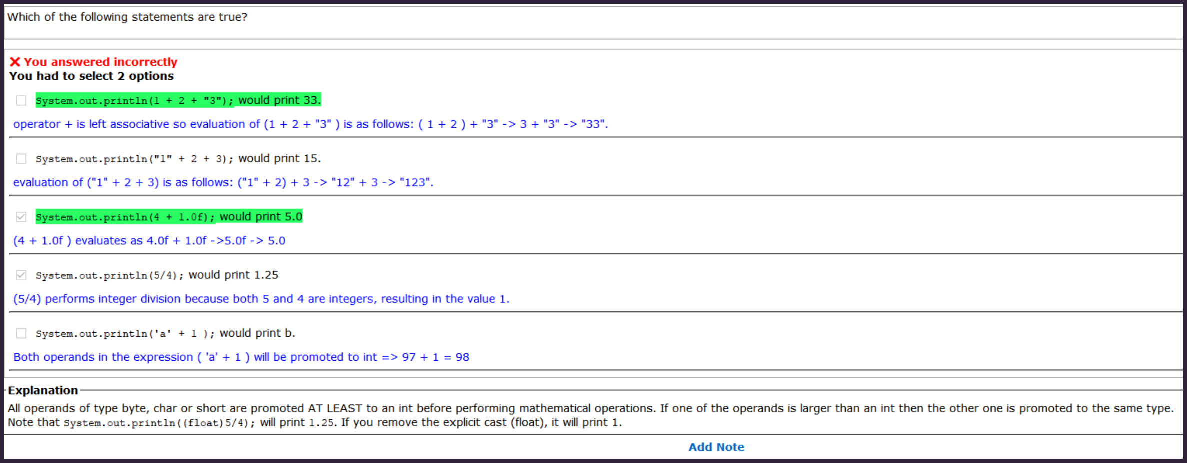
Since the first one is numeric type so, 'b'+63 = 161, 161+"a" = 161a.

[ ] `String s = 63 + new Integer(10);`

Since neither of the operands of + operator is a String, it will not generate a String. However, due to auto-unboxing of 10, it will generate an int value of 73.

Explanation

+ is overloaded such that if any one of its two operands is a String then it will convert the other operand to a String and create a new string by concatenating the two.
Therefore, in 63+"a" and "a"+63, 63 is converted to "63" and 'b' +"a" and "a"+'b', 'b' is converted to "b".
Note that in 'b'+ 63 , 'b' is promoted to an int i.e. 98 giving 161.



Which of the following statements are true?

✗ You answered incorrectly
You had to select 2 options

[ ] `System.out.println(1 + 2 + "3");` would print 33.

operator + is left associative so evaluation of (1 + 2 + "3") is as follows: (1 + 2) + "3" -> 3 + "3" -> "33".

[ ] `System.out.println("1" + 2 + 3);` would print 15.

evaluation of ("1" + 2 + 3) is as follows: ("1" + 2) + 3 -> "12" + 3 -> "123".

[✓] `System.out.println(4 + 1.0f);` would print 5.0

(4 + 1.0f) evaluates as 4.0f + 1.0f ->5.0f -> 5.0

[✓] `System.out.println(5/4);` would print 1.25

(5/4) performs integer division because both 5 and 4 are integers, resulting in the value 1.

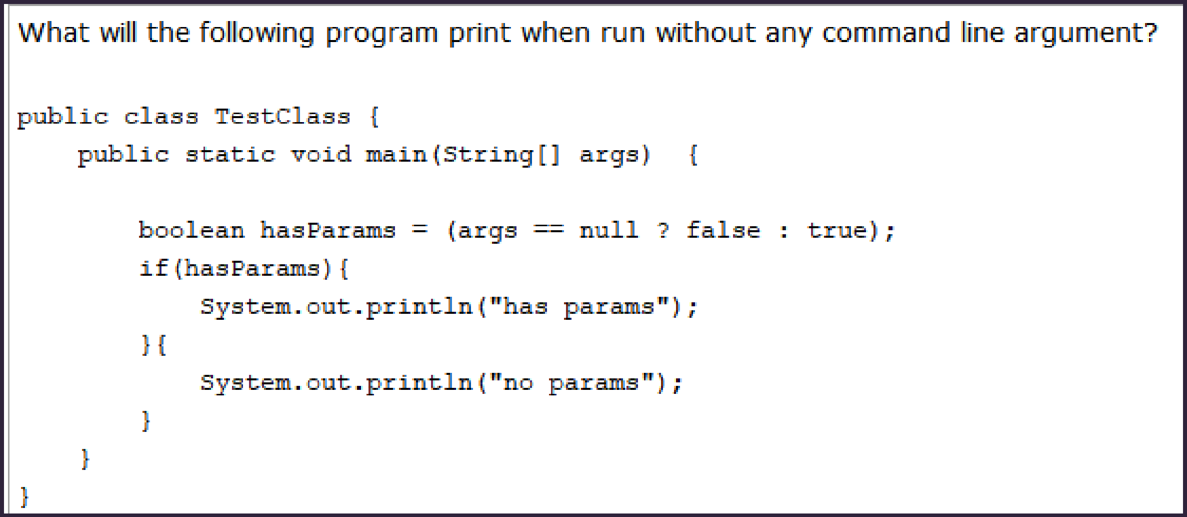
[ ] `System.out.println('a' + 1 );` would print b.

Both operands in the expression ('a' + 1) will be promoted to int => 97 + 1 = 98

Explanation

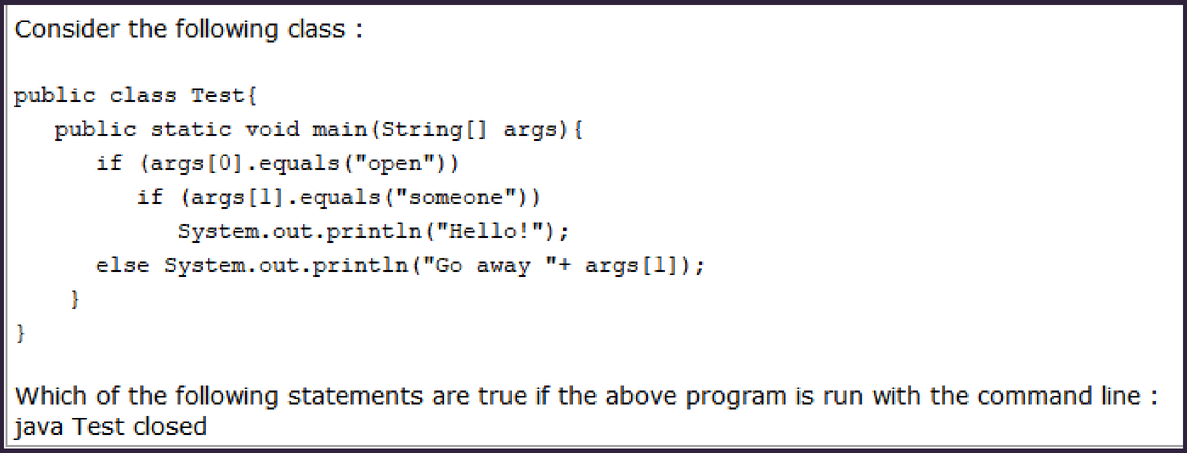
All operands of type byte, char or short are promoted AT LEAST to an int before performing mathematical operations. If one of the operands is larger than an int then the other one is promoted to the same type.
Note that `System.out.println((float)5/4);` will print `1.25`. If you remove the explicit cast (float), it will print 1.

Add Note



What will the following program print when run without any command line argument?

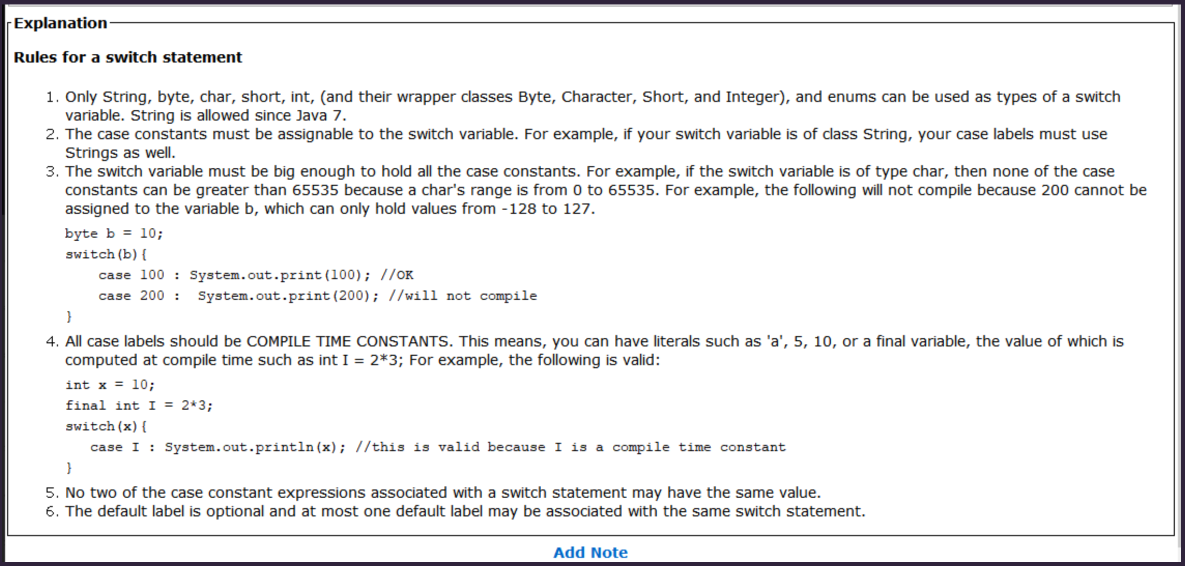
```
public class TestClass {
    public static void main(String[] args)   {

        boolean hasParams = (args == null ? false : true);
        if(hasParams){
            System.out.println("has params");
        }{
            System.out.println("no params");
        }
    }
}
```



Consider the following class :

```
public class Test{
   public static void main(String[] args){
      if (args[0].equals("open"))
         if (args[1].equals("someone"))
            System.out.println("Hello!");
      else System.out.println("Go away "+ args[1]);
    }
}
```

Which of the following statements are true if the above program is run with the command line : java Test closed



Explanation

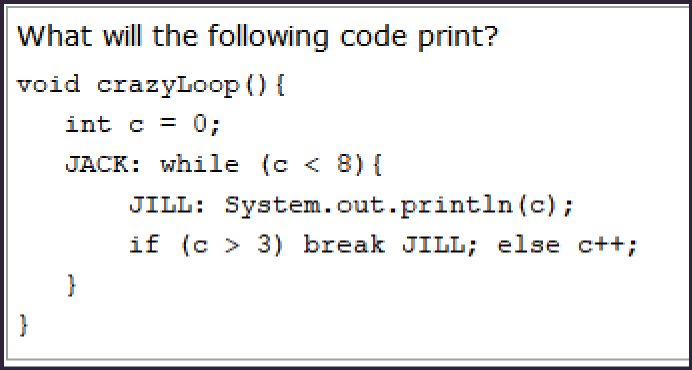
Rules for a switch statement

1. Only String, byte, char, short, int, (and their wrapper classes Byte, Character, Short, and Integer), and enums can be used as types of a switch variable. String is allowed since Java 7.
2. The case constants must be assignable to the switch variable. For example, if your switch variable is of class String, your case labels must use Strings as well.
3. The switch variable must be big enough to hold all the case constants. For example, if the switch variable is of type char, then none of the case constants can be greater than 65535 because a char's range is from 0 to 65535. For example, the following will not compile because 200 cannot be assigned to the variable b, which can only hold values from -128 to 127.
   ```
   byte b = 10;
   switch(b){
       case 100 : System.out.print(100); //OK
       case 200 :  System.out.print(200); //will not compile
   }
   ```
4. All case labels should be COMPILE TIME CONSTANTS. This means, you can have literals such as 'a', 5, 10, or a final variable, the value of which is computed at compile time such as int I = 2*3; For example, the following is valid:
   ```
   int x = 10;
   final int I = 2*3;
   switch(x){
      case I : System.out.println(x); //this is valid because I is a compile time constant
   }
   ```
5. No two of the case constant expressions associated with a switch statement may have the same value.
6. The default label is optional and at most one default label may be associated with the same switch statement.

Add Note

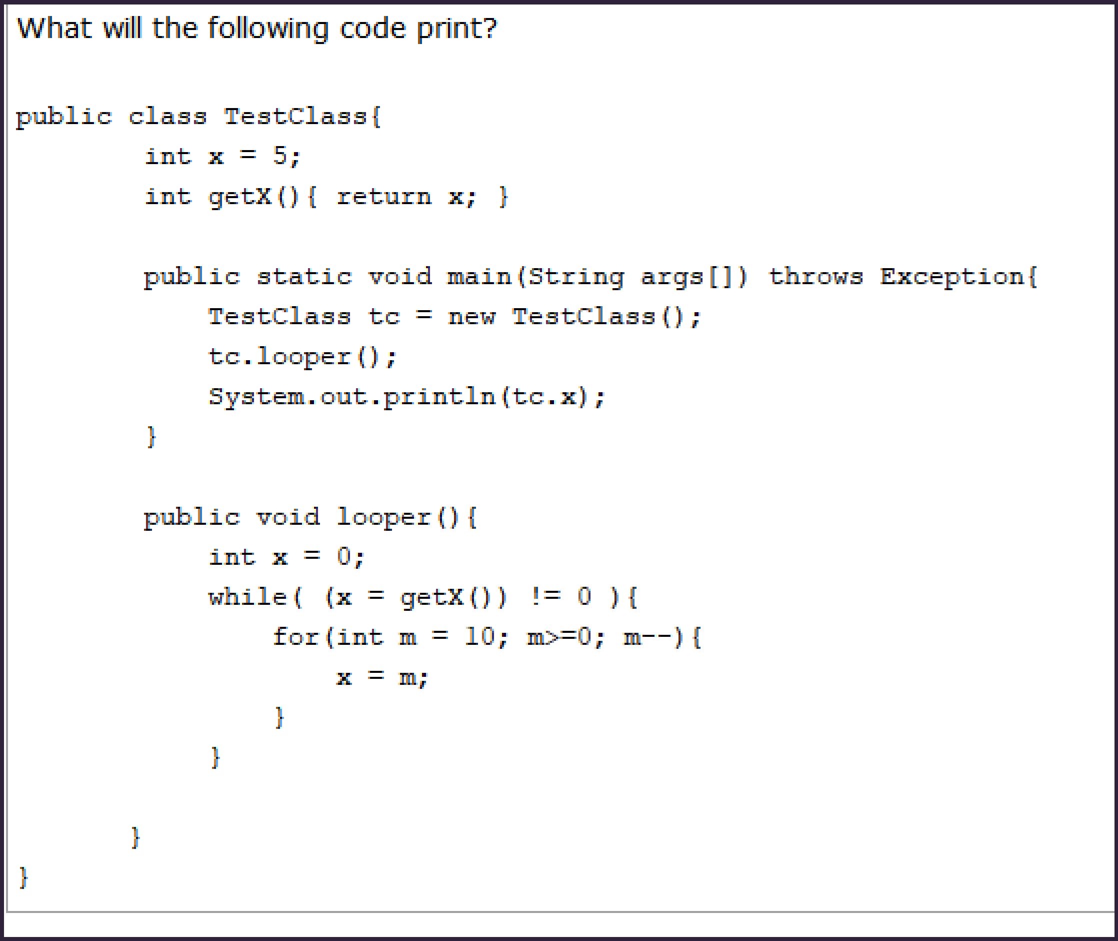
04-

05-USING LOOP CONSTRUCTORS



What will the following code print?

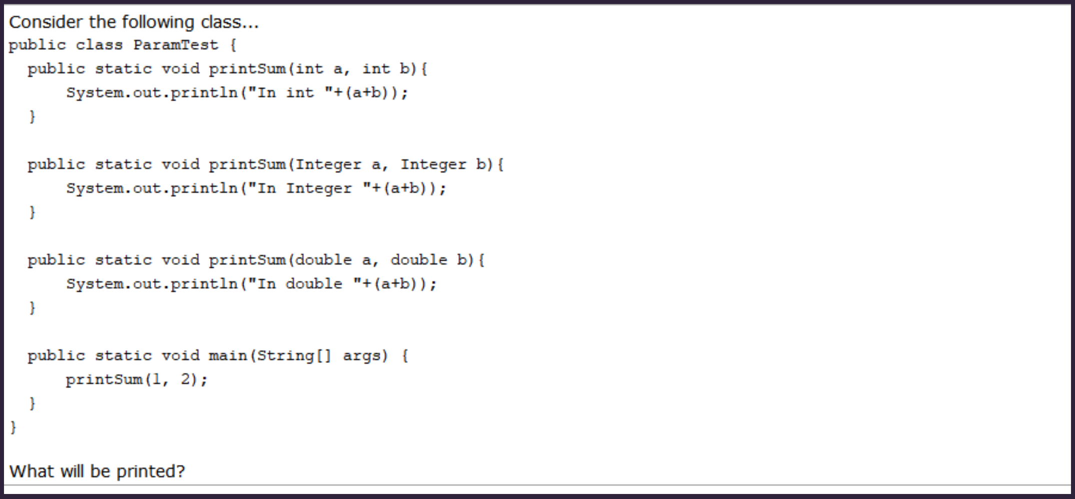
```
void crazyLoop(){
   int c = 0;
   JACK: while (c < 8){
       JILL: System.out.println(c);
       if (c > 3) break JILL; else c++;
   }
}
```



What will the following code print?

```
public class TestClass{
        int x = 5;
        int getX(){ return x; }

        public static void main(String args[]) throws Exception{
            TestClass tc = new TestClass();
            tc.looper();
            System.out.println(tc.x);
        }

        public void looper(){
            int x = 0;
            while( (x = getX()) != 0 ){
                for(int m = 10; m>=0; m--){
                    x = m;
                }
            }

       }
}
```

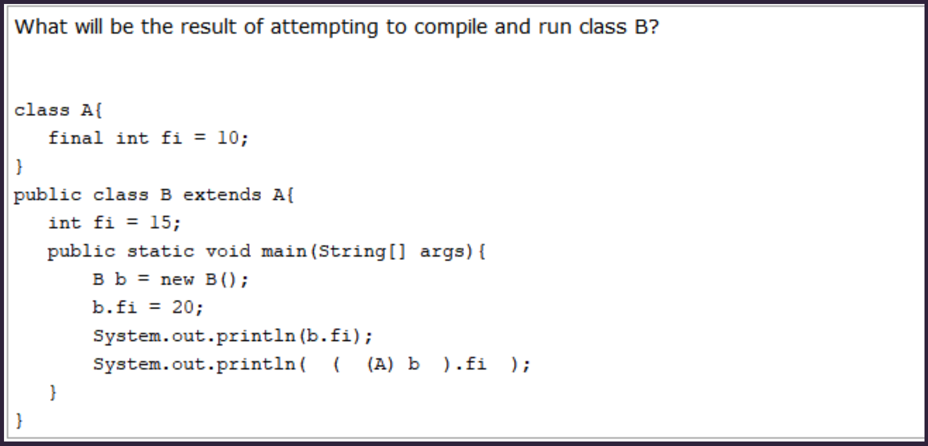
06-WORKING WITH METHODS OVERLOADING


Consider the following class...

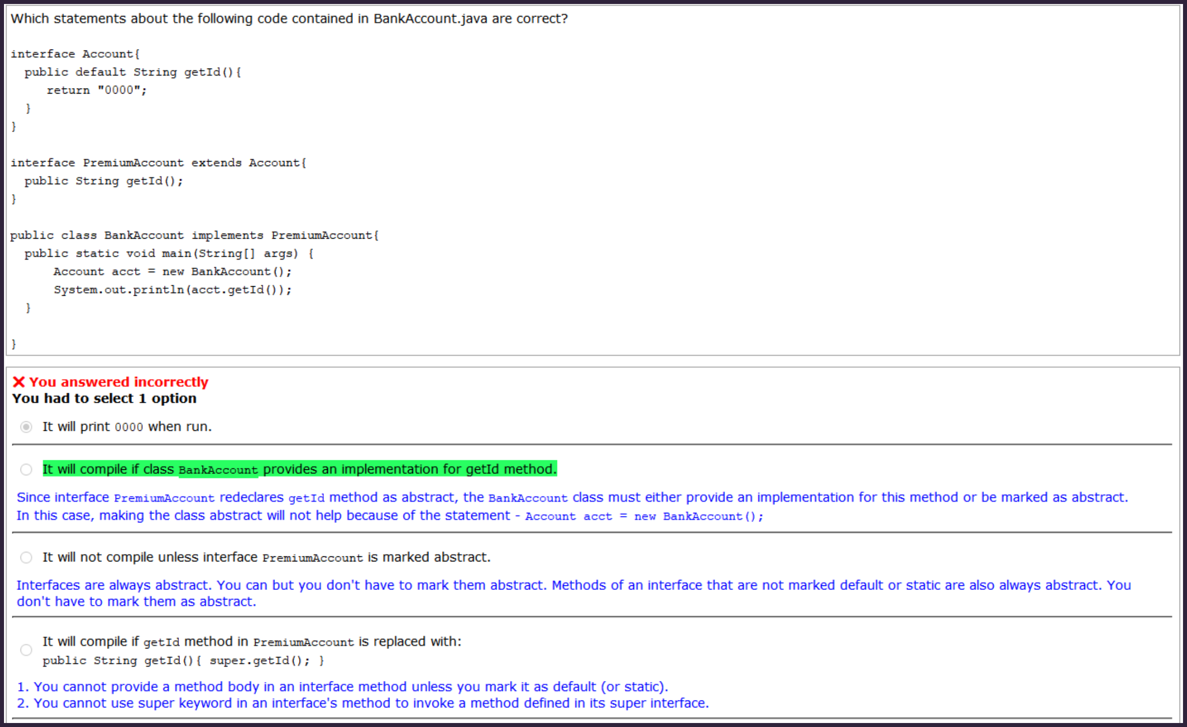
```
public class ParamTest {
  public static void printSum(int a, int b){
      System.out.println("In int "+(a+b));
  }

  public static void printSum(Integer a, Integer b){
      System.out.println("In Integer "+(a+b));
  }

  public static void printSum(double a, double b){
      System.out.println("In double "+(a+b));
  }

  public static void main(String[] args) {
      printSum(1, 2);
  }
}
```

What will be printed?

07-WORKING WITH INHERITANCE


What will be the result of attempting to compile and run class B?

```
class A{
   final int fi = 10;
}
public class B extends A{
   int fi = 15;
   public static void main(String[] args){
       B b = new B();
       b.fi = 20;
       System.out.println(b.fi);
       System.out.println(  (  (A) b  ).fi  );
   }
}
```



Which statements about the following code contained in BankAccount.java are correct?

```
interface Account{
  public default String getId(){
     return "0000";
  }
}

interface PremiumAccount extends Account{
  public String getId();
}

public class BankAccount implements PremiumAccount{
  public static void main(String[] args) {
      Account acct = new BankAccount();
      System.out.println(acct.getId());
  }

}
```

✗ You answered incorrectly
You had to select 1 option

[✓ It will print `0000` when run.]

It will compile if class `BankAccount` provides an implementation for getId method.

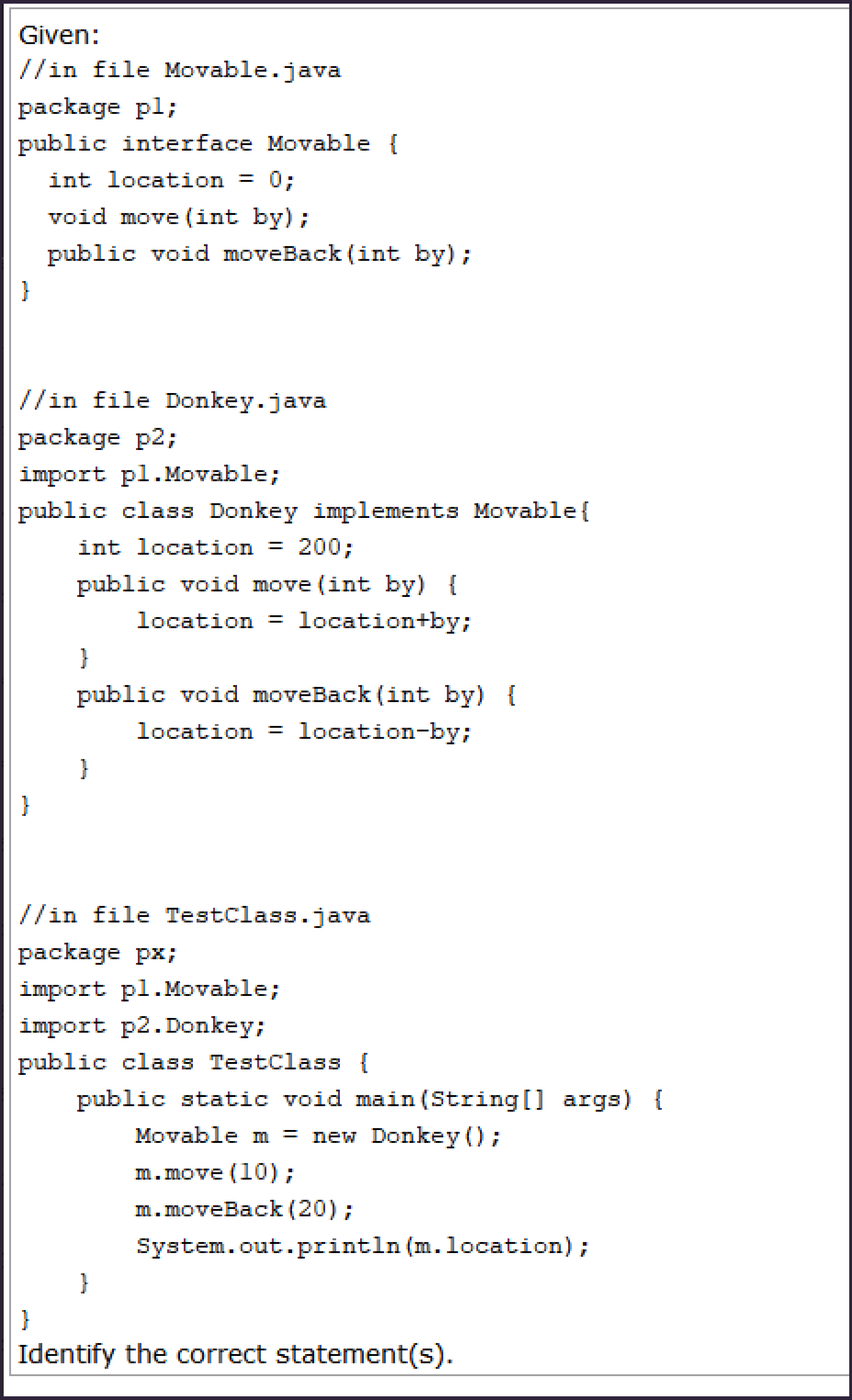
Since interface `PremiumAccount` redeclares `getId` method as abstract, the `BankAccount` class must either provide an implementation for this method or be marked as abstract.
In this case, making the class abstract will not help because of the statement - `Account acct = new BankAccount();`

It will not compile unless interface `PremiumAccount` is marked abstract.

Interfaces are always abstract. You can but you don't have to mark them abstract. Methods of an interface that are not marked default or static are also always abstract. You don't have to mark them as abstract.

It will compile if `getId` method in `PremiumAccount` is replaced with:
`public String getId(){ super.getId(); }`

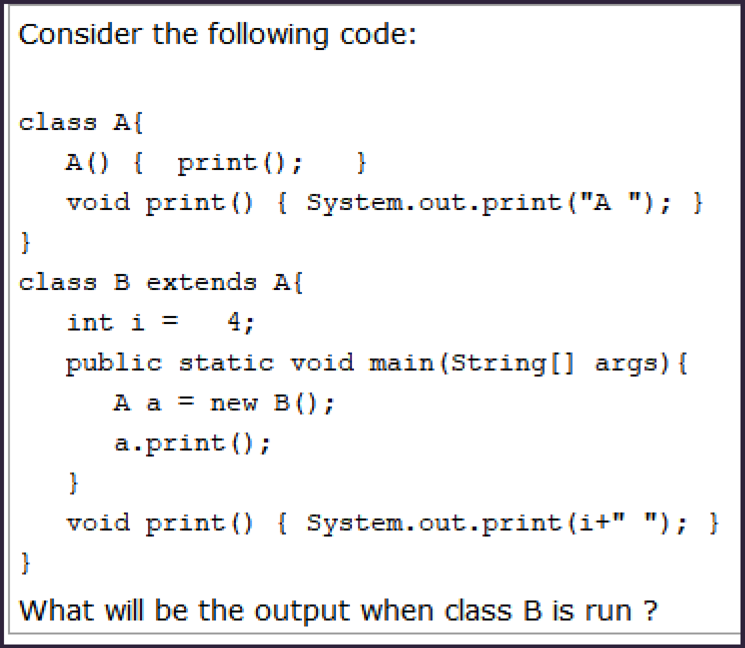
1. You cannot provide a method body in an interface method unless you mark it as default (or static).
2. You cannot use super keyword in an interface's method to invoke a method defined in its super interface.



Given:

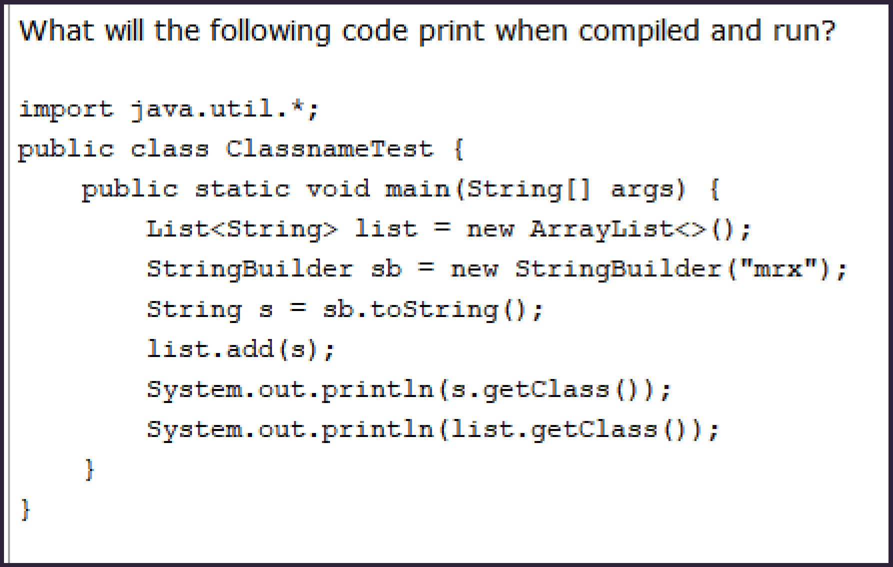
```
//in file Movable.java
package pl;
public interface Movable {
  int location = 0;
  void move(int by);
  public void moveBack(int by);
}


//in file Donkey.java
package p2;
import pl.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 pl.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);
    }
}
```

Identify the correct statement(s).


Consider the following code:

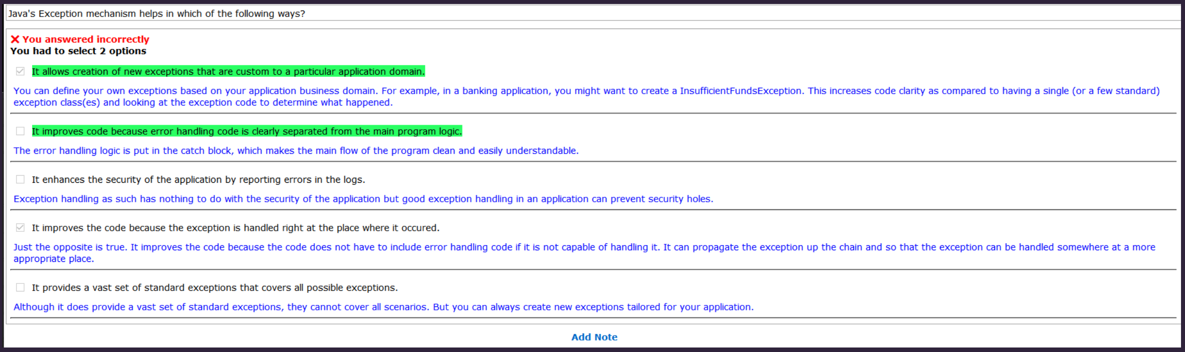
```
class A{
   A() {  print();   }
   void print() { System.out.print("A "); }
}
class B extends A{
   int i =   4;
   public static void main(String[] args){
      A a = new B();
      a.print();
   }
   void print() { System.out.print(i+" "); }
}
```

What will be the output when class B is run ?


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());
    }
}
```

08- HANDLING EXCEPTIONS



Java's Exception mechanism helps in which of the following ways?

✗ You answered incorrectly
You had to select 2 options

- [x] It allows creation of new exceptions that are custom to a particular application domain.

You can define your own exceptions based on your application business domain. For example, in a banking application, you might want to create a InsufficientFundsException. This increases code clarity as compared to having a single (or a few standard) exception class(es) and looking at the exception code to determine what happened.

- [ ] It improves code because error handling code is clearly separated from the main program logic.

The error handling logic is put in the catch block, which makes the main flow of the program clean and easily understandable.

- [ ] It enhances the security of the application by reporting errors in the logs.

Exception handling as such has nothing to do with the security of the application but good exception handling in an application can prevent security holes.

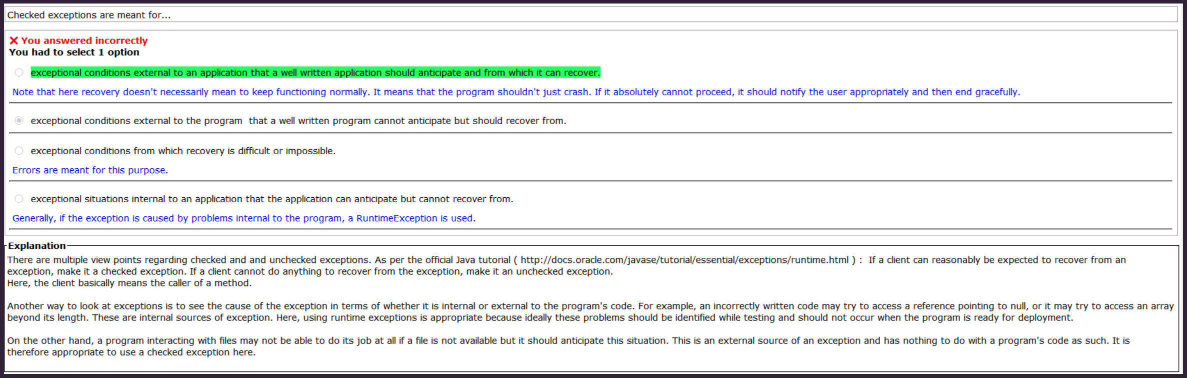
- [x] It improves the code because the exception is handled right at the place where it occured.

Just the opposite is true. It improves the code because the code does not have to include error handling code if it is not capable of handling it. It can propagate the exception up the chain and so that the exception can be handled somewhere at a more appropriate place.

- [ ] It provides a vast set of standard exceptions that covers all possible exceptions.

Although it does provide a vast set of standard exceptions, they cannot cover all scenarios. But you can always create new exceptions tailored for your application.

Add Note



Checked exceptions are meant for...

✗ You answered incorrectly
You had to select 1 option

○ exceptional conditions external to an application that a well written application should anticipate and from which it can recover.

Note that here recovery doesn't necessarily mean to keep functioning normally. It means that the program shouldn't just crash. If it absolutely cannot proceed, it should notify the user appropriately and then end gracefully.

● exceptional conditions external to the program that a well written program cannot anticipate but should recover from.

○ exceptional conditions from which recovery is difficult or impossible.

Errors are meant for this purpose.

○ exceptional situations internal to an application that the application can anticipate but cannot recover from.

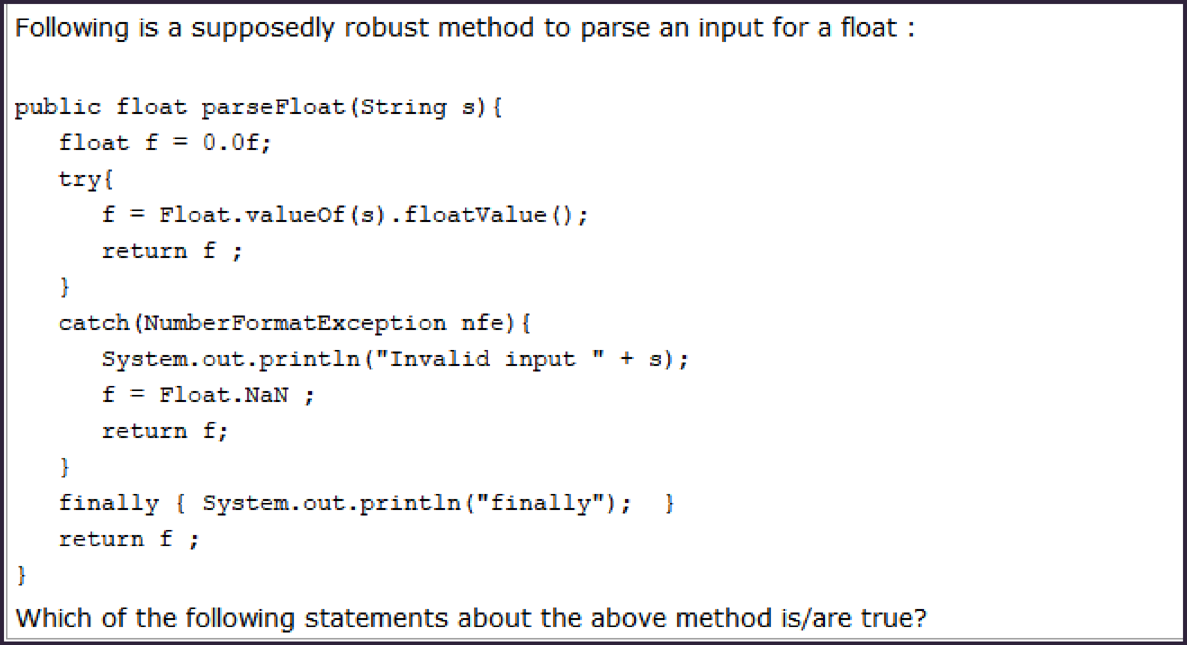
Generally, if the exception is caused by problems internal to the program, a RuntimeException is used.

Explanation

There are multiple view points regarding checked and and unchecked exceptions. As per the official Java tutorial (http://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html) : If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception.
Here, the client basically means the caller of a method.

Another way to look at exceptions is to see the cause of the exception in terms of whether it is internal or external to the program's code. For example, an incorrectly written code may try to access a reference pointing to null, or it may try to access an array beyond its length. These are internal sources of exception. Here, using runtime exceptions is appropriate because ideally these problems should be identified while testing and should not occur when the program is ready for deployment.

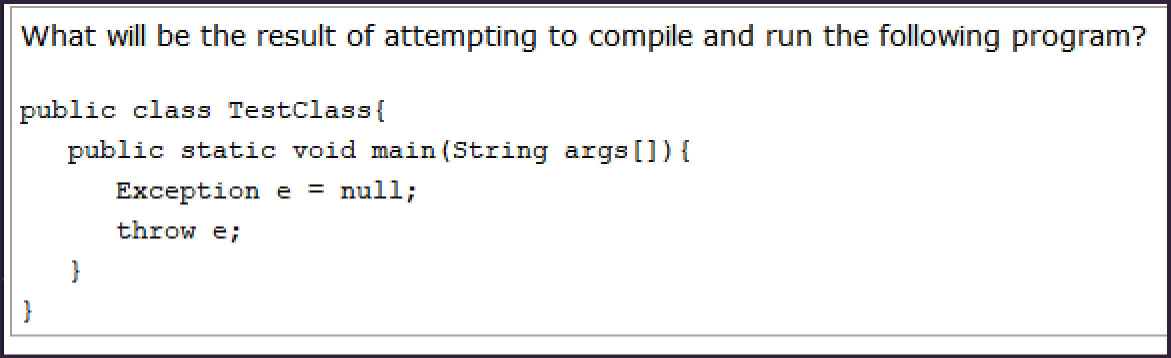
On the other hand, a program interacting with files may not be able to do its job at all if a file is not available but it should anticipate this situation. This is an external source of an exception and has nothing to do with a program's code as such. It is therefore appropriate to use a checked exception here.



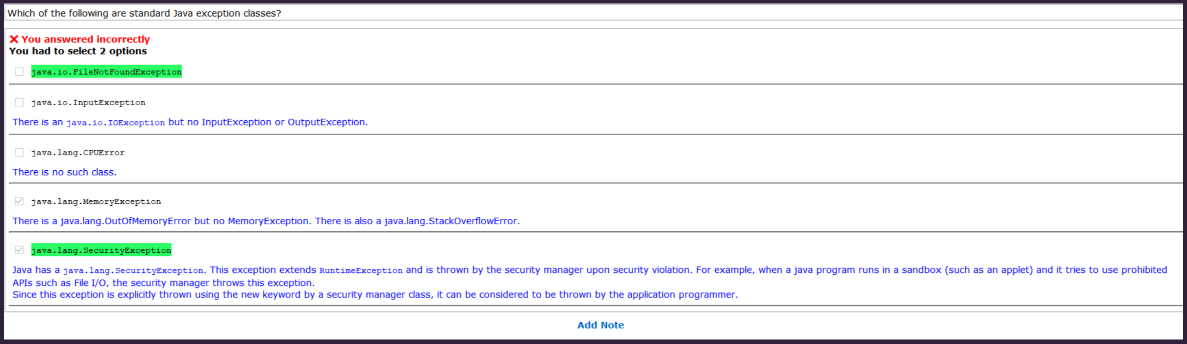
Following is a supposedly robust method to parse an input for a float :

```
public float parseFloat(String s){
   float f = 0.0f;
   try{
      f = Float.valueOf(s).floatValue();
      return f ;
   }
   catch(NumberFormatException nfe){
      System.out.println("Invalid input " + s);
      f = Float.NaN ;
      return f;
   }
   finally { System.out.println("finally");   }
   return f ;
}
```

Which of the following statements about the above method is/are true?


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

```
public class TestClass{
   public static void main(String args[]){
      Exception e = null;
      throw e;
   }
}
```


Which of the following are standard Java exception classes?

✗ You answered incorrectly
You had to select 2 options

- [ ] `java.io.FileNotFoundException`
- [ ] `java.io.InputException`

There is an `java.io.IOException` but no InputException or OutputException.

- [ ] `java.lang.CPUError`

There is no such class.

- [✓] `java.lang.MemoryException`

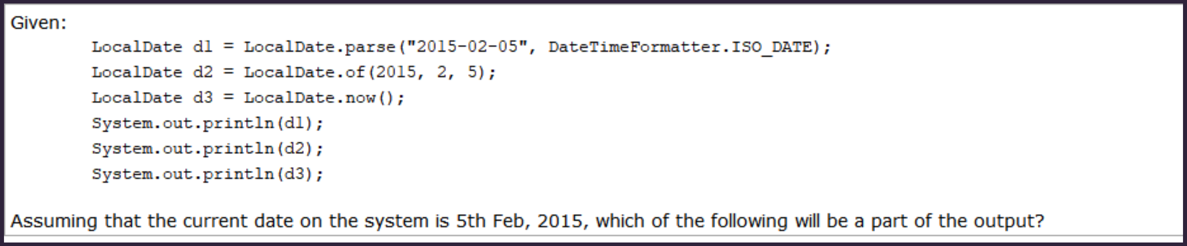
There is a java.lang.OutOfMemoryError but no MemoryException. There is also a java.lang.StackOverflowError.

- [✓] `java.lang.SecurityException`

Java has a `java.lang.SecurityException`. This exception extends `RuntimeException` and is thrown by the security manager upon security violation. For example, when a java program runs in a sandbox (such as an applet) and it tries to use prohibited APIs such as File I/O, the security manager throws this exception.
Since this exception is explicitly thrown using the new keyword by a security manager class, it can be considered to be thrown by the application programmer.

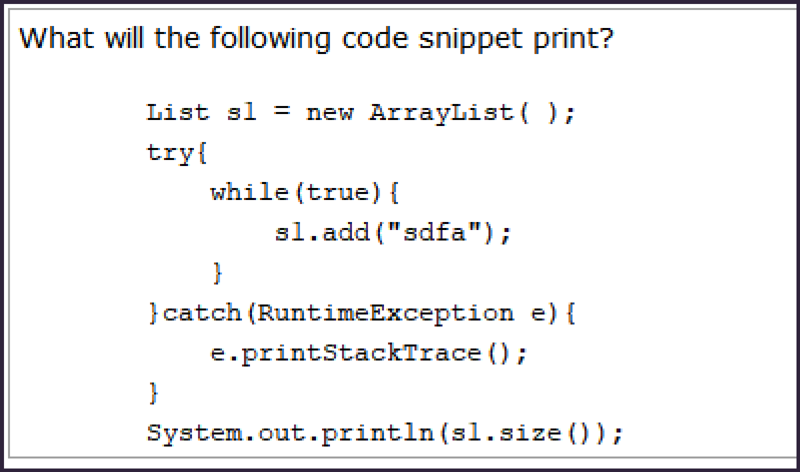
Add Note

09- WORKING WITH JAVA API


Given:

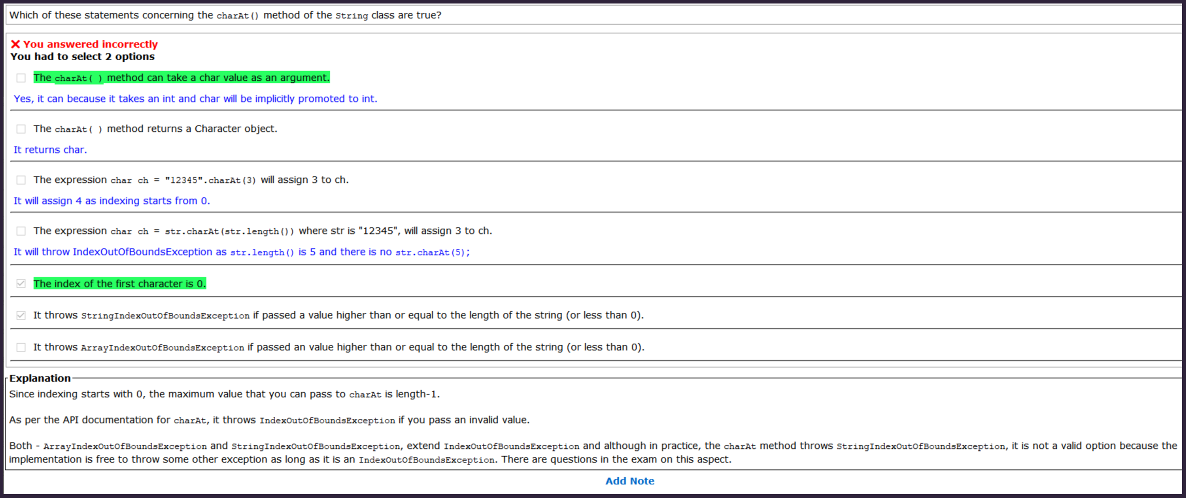
```
        LocalDate d1 = LocalDate.parse("2015-02-05", DateTimeFormatter.ISO_DATE);
        LocalDate d2 = LocalDate.of(2015, 2, 5);
        LocalDate d3 = LocalDate.now();
        System.out.println(d1);
        System.out.println(d2);
        System.out.println(d3);
```

Assuming that the current date on the system is 5th Feb, 2015, which of the following will be a part of the output?


What will the following code snippet print?

```
        List sl = new ArrayList( );
        try{
            while(true){
                sl.add("sdfa");
            }
        }catch(RuntimeException e){
            e.printStackTrace();
        }
        System.out.println(sl.size());
```

You want to find out whether two strings are equal or not, in terms of the actual characters within the strings. What is the best way to do this?



Which of these statements concerning the `charAt()` method of the `String` class are true?

✗ You answered incorrectly
You had to select 2 options

The `charAt( )` method can take a char value as an argument.

Yes, it can because it takes an int and char will be implicitly promoted to int.

The `charAt( )` method returns a Character object.

It returns char.

The expression `char ch = "12345".charAt(3)` will assign 3 to ch.

It will assign 4 as indexing starts from 0.

The expression `char ch = str.charAt(str.length())` where str is "12345", will assign 3 to ch.

It will throw IndexOutOfBoundsException as `str.length()` is 5 and there is no `str.charAt(5);`

[✓ The index of the first character is 0.]

[✓ It throws `StringIndexOutOfBoundsException` if passed a value higher than or equal to the length of the string (or less than 0).]

It throws `ArrayIndexOutOfBoundsException` if passed an value higher than or equal to the length of the string (or less than 0).

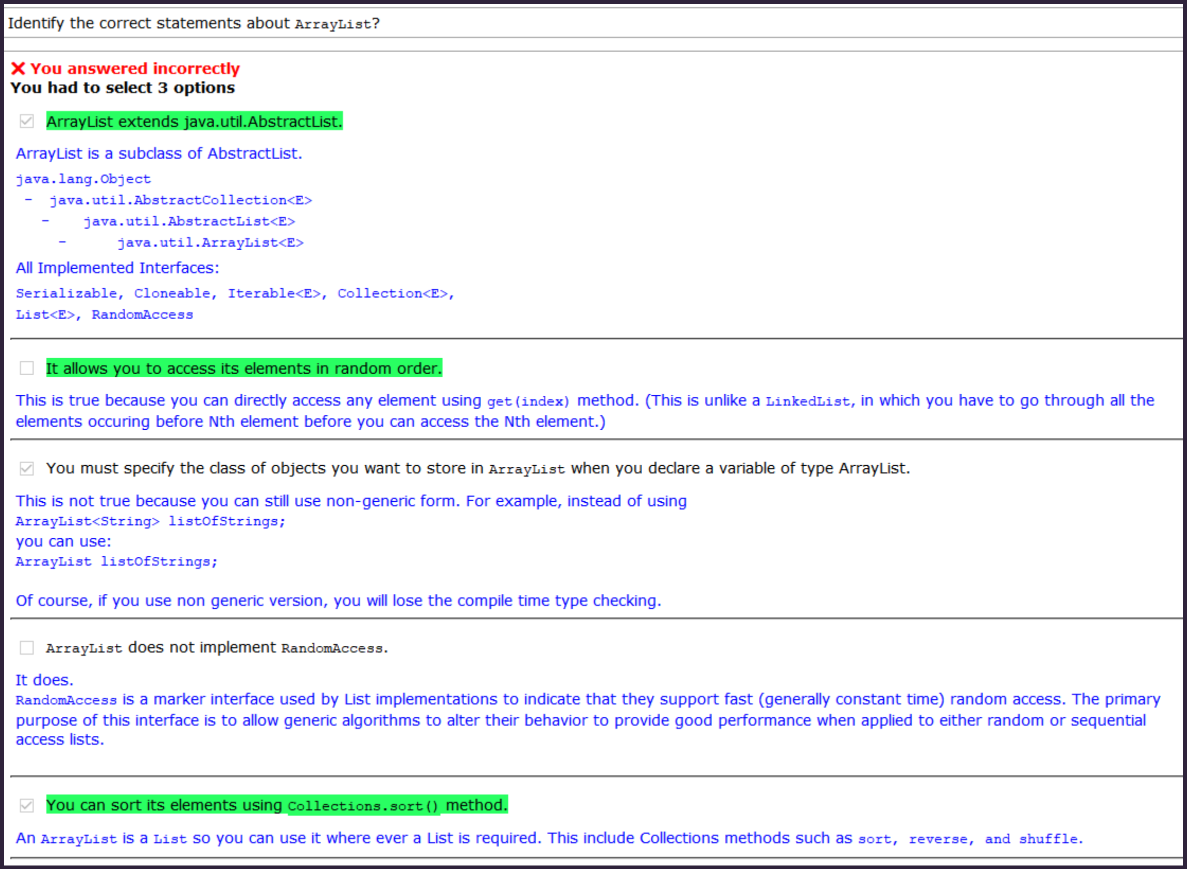
Explanation

Since indexing starts with 0, the maximum value that you can pass to `charAt` is length-1.

As per the API documentation for `charAt`, it throws `IndexOutOfBoundsException` if you pass an invalid value.

Both - `ArrayIndexOutOfBoundsException` and `StringIndexOutOfBoundsException`, extend `IndexOutOfBoundsException` and although in practice, the `charAt` method throws `StringIndexOutOfBoundsException`, it is not a valid option because the implementation is free to throw some other exception as long as it is an `IndexOutOfBoundsException`. There are questions in the exam on this aspect.

Add Note



Identify the correct statements about `ArrayList`?

✗ You answered incorrectly
You had to select 3 options

[✓ ArrayList extends java.util.AbstractList.]

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`

It allows you to access its elements in random order.

This is true because you can directly access any element using `get(index)` method. (This is unlike a `LinkedList`, in which you have to go through all the elements occuring before Nth element before you can access the Nth element.)

[✓ You must specify the class of objects you want to store in `ArrayList` when you declare a variable of type ArrayList.]

This is not true because you can still use non-generic form. For example, instead of using
`ArrayList<String> listOfStrings;`
you can use:
`ArrayList listOfStrings;`

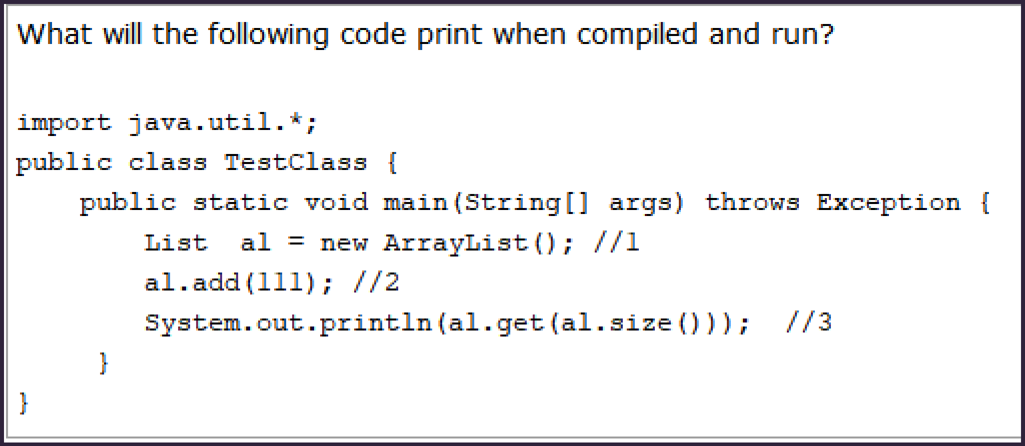
Of course, if you use non generic version, you will lose the compile time type checking.

`ArrayList` does not implement `RandomAccess`.

It does.
`RandomAccess` is a marker interface used by List implementations to indicate that they support fast (generally constant time) random access. The primary purpose of this interface is to allow generic algorithms to alter their behavior to provide good performance when applied to either random or sequential access lists.

[✓ You can sort its elements using `Collections.sort()` method.]

An `ArrayList` is a `List` so you can use it where ever a List is required. This include Collections methods such as `sort, reverse, and shuffle`.



What will the following code print when compiled and run?

```
import java.util.*;
public class TestClass {
    public static void main(String[] args) throws Exception {
        List  al = new ArrayList(); //1
        al.add(111); //2
        System.out.println(al.get(al.size()));  //3
     }
}
```