02 Java
Using Operators & Decision Constructs

Problema 1

Consider the following lines of code:

```
boolean greenLight = true;

boolean pedestrian = false;

boolean rightTurn = true;

boolean otherLane = false;
```

You can go ahead only if the following expression evaluates to 'true' :

```
(( (rightTurn && !pedestrian || otherLane)
       || ( ? && !pedestrian && greenLight ) )  == true )
```

What variables can you put in place of '?' so that you can go ahead?

You had to select 1 option

- rightTurn
- otherLane
- [✓ Any variable would do.]

  since the part before second || is true, the next part is not even evaluated.
- None of the variable would allow to go.

Explanation:

Observe that (`rightTurn && !pedestrian || otherLane`) is true, therefore (`? && !pedestrian && greenLight`) does not matter.

|| and && are short circuit operators. So, if the first part of the expression (i.e. part before ||) is true (or false for &&) the other part is not evaluated at all.

Note that this is not true for | and &. In that case, the whole expression will be evaluated even if the value of the expression can be known by just evaluating first part.

Problema 2

What will the following code print when run?

```
public class TestClass {
    public void switchString(String input){
        switch(input){
            case "a" : System.out.println( "apple" );
            case "b" : System.out.println( "bat" );
                break;
            case "B" : System.out.println( "big bat" );
            default : System.out.println( "none" );
        }
    }

    public static void main(String[] args) throws Exception {
        TestClass tc = new TestClass();
        tc.switchString("B");
    }
}
```

- ○ bat
  big bat
- ○ big bat
  none

Since there is a case condition that matches the input string "B", that case statement will be executed directly. This prints "big bat". Since there is no break after this case statement and the next case statement, the control will fall through the next one (which is default :) and so "none" will be printed as well.

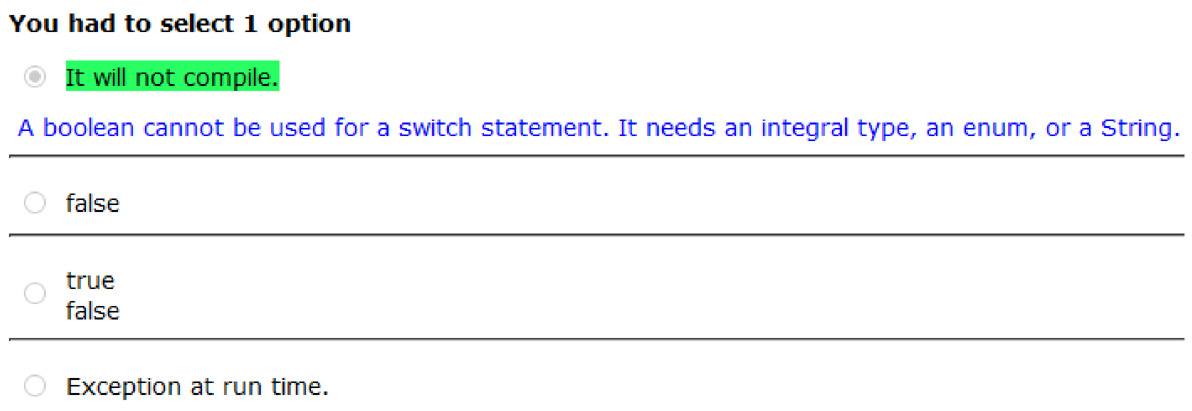
Note that "b" and "B" are different strings. "B" is not equal to "b".

- [✓ big bat]
- ○ bat
- ○ The code will not compile.

Problema 3

What will the following code print when run?

```
public class TestClass {

    public static void main(String[] args) throws Exception {


         boolean flag  = true;

         switch (flag){

             case true : System.out.println("true");

                 default: System.out.println("false");

         }


    }

}
```



You had to select 1 option

- [✓ It will not compile.]

  A boolean cannot be used for a switch statement. It needs an integral type, an enum, or a String.

- false

- true
  false

- Exception at run time.

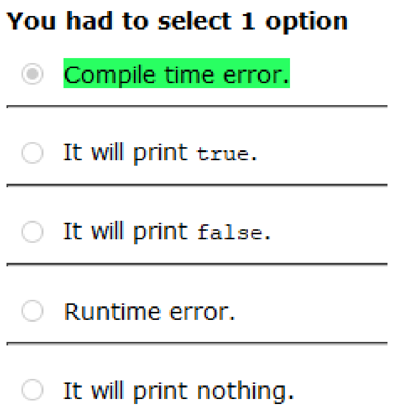
Explanation

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.

Problema 4

What is the result of executing the following fragment of code:

```
boolean b1 = false;

boolean b2 = false;

if (b2 != b1 = !b2){

   System.out.println("true");

}

else{

   System.out.println("false");

}
```



You had to select 1 option

- [✓ Compile time error.]
- It will print `true`.
- It will print `false`.
- Runtime error.
- It will print nothing.

Explanation

Note that boolean operators have more precedence than =. (In fact, = has least precedence of all operators.)

so, in (b2 != b1 = !b2) first b2 != b1 is evaluated which returns a value 'false'. So the expression becomes false = !b2. And this is illegal because false is a value and not a variable!

Had it been something like (b2 = b1 != b2) then it is valid because it will boil down to : b2 = false.

Because all an if() needs is a boolean, now b1 != b2 returns false which is a boolean and as b2 = false is an expression and every expression has a return value (which is actually the Left Hand Side of the expression). Here, it returns false, which is again a boolean.

Problema 5

What can be the return type of method getSwitch so that this program compiles and runs without any problems?

```
public class TestClass{
   public static XXX getSwitch(int x){
      return x - 20/x + x*x;
   }
   public static void main(String args[]){
       switch( getSwitch(10) ){
          case 1 :
          case 2 :
          case 3 :
          default : break;
       }
   }
}
```

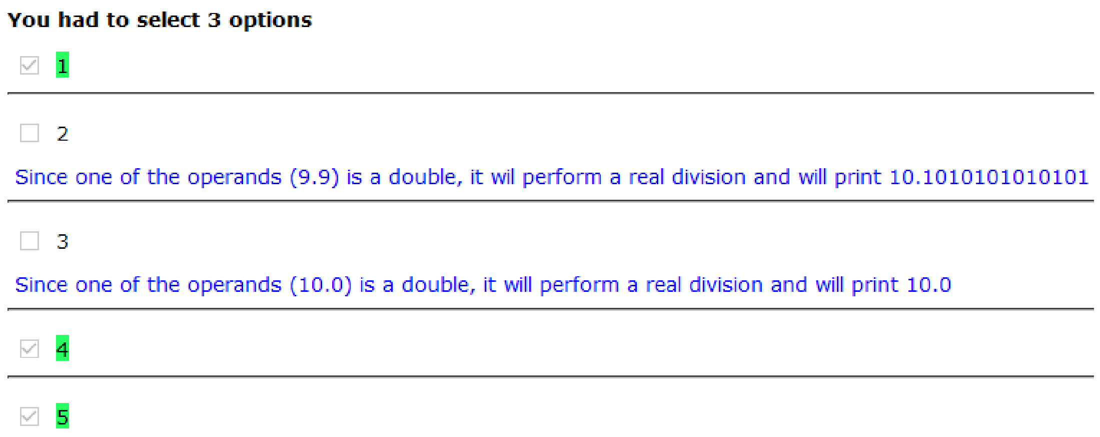
Explanation:

If you just consider the method getSwitch, any of int long float or double will do. But the return value is used in the switch statement later on. A switch condition cannot accept float, long, double, or boolean. So only int is valid. The return type cannot be byte, short, or char because the expression x - 20/x + x*x; returns an int.

Problema 6

Which of the following code snippets will print exactly 10?

```
1.  Object t = new Integer(106);

    int k = ((Integer) t).intValue()/10;

    System.out.println(k);


2.  System.out.println(100/9.9);


3.  System.out.println(100/10.0);


4.  System.out.println(100/10);


5.  System.out.println(3 + 100/10*2-13);
```



You had to select 3 options

[✓ 1]

☐ 2

Since one of the operands (9.9) is a double, it wil perform a real division and will print 10.1010101010101

☐ 3

Since one of the operands (10.0) is a double, it will perform a real division and will print 10.0

[✓ 4]

[✓ 5]

```
1. int k = ((Integer) t).intValue()/10;
```

Since both the operands of / are ints, it is a integer division. This means the resulting value is truncated (and not rounded). Therefore, the above statement will print 10 and not 11. 5. 3 + 100/10*2-13 will be parsed as: 3 + (100/10)*2-13. This is because the precedence of / and * is same (and is higher than + and -) and since the expression is evaluated from left to right, the operands are grouped on first come first served basis. [This is not the right terminology but you will be able to answer the questions if you remember this rule.]

Problema 7

The following method will compile and run without any problems.

```
public void switchTest(byte x){
   switch(x){
      case 'b':   // 1
      default :   // 2
      case -2:    // 3
      case 80:    // 4
   }
}
```

You had to select 1 option

[✓ True]

False

Explanation:

The following types can be used as a switch variable:

byte, char, short, int, String, and enums. Wrapper classes Byte, Character, Short, and Integer are allowed as well. Note that long, float, double, and boolean are not allowed.

All the case constants should be assignable to the switch variable type. i.e. had there been a case label of 128 (case 128 : //some code), it would not have compiled. Because the range of a byte is from -128 to 127 and so 128 is not assignable to 'x'.

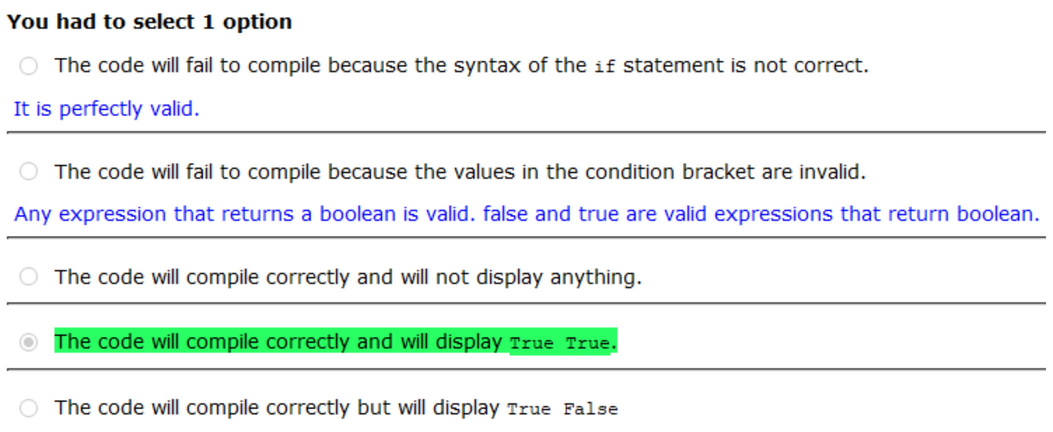
The integral value of 'b' is 98, which is less than 127 so Line //1 is fine.

Note: Although it is not required for the exam to know the integral values of characters, it is good to know that all English letters (upper case as well as lower case) as well as 0-9 are below 127 and so are assignable to byte.

Problema 8

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

```
public class IfTest{

   public static void main(String args[]){

      if (true)

      if (false)

      System.out.println("True False");

      else

      System.out.println("True True");

   }

}
```



You had to select 1 option

- The code will fail to compile because the syntax of the `if` statement is not correct.

  It is perfectly valid.

- The code will fail to compile because the values in the condition bracket are invalid.

  Any expression that returns a boolean is valid. false and true are valid expressions that return boolean.

- The code will compile correctly and will not display anything.

- [✓ The code will compile correctly and will display `True True`.]

- The code will compile correctly but will display `True False`

Explanation:

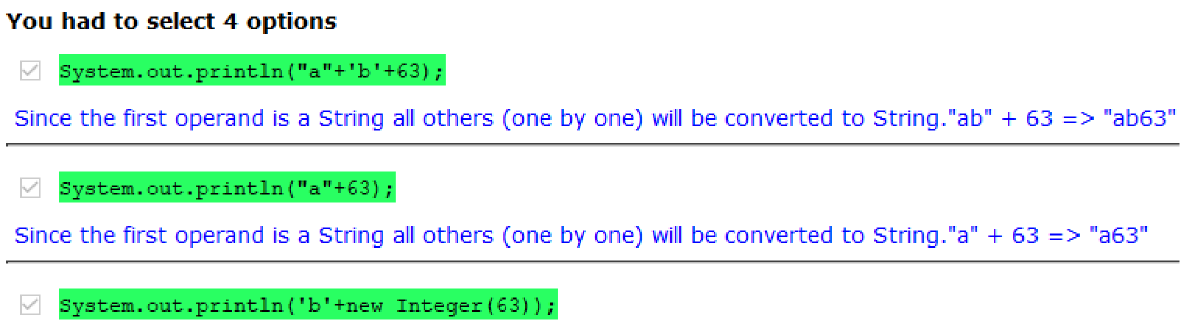
This code can be rewritten as follows:

```
public class IfTest{
    public static void main(String args[]) {
        if (true) {
            if (false) {
                System.out.println("True False");
            } else {
                System.out.println("True True");
            }
        }
    }
}
```

Notice how the last "else" is associated with the last "if" and not the first "if". Now, the first if condition returns true so the next 'if' will be executed. In the second 'if' the condition returns false so the else part will be evaluated which prints 'True True'.

Problema 9

Which of the following statements will compile without any error?



You had to select 4 options

✓ `System.out.println("a"+'b'+63);`

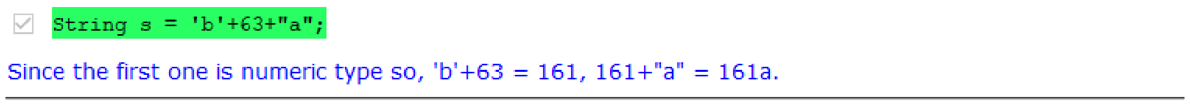
Since the first operand is a String all others (one by one) will be converted to String."ab" + 63 => "ab63"

✓ `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"

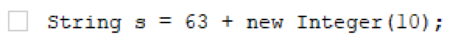
✓ `System.out.println('b'+new Integer(63));`

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".

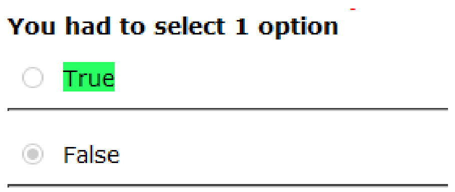
Note that in 'b'+ 63 , 'b' is promoted to an int i.e. 98 giving 161.

Problema 10

The following code snippet will not compile:

```
int i = 10;

System.out.println( i<20 ? out1() : out2() );
```

Assume that out1 and out2 methods have the following signatures: public void out1(); and public void out2();



You had to select 1 option

True [✓ False]

Explanation:

Note that it is not permitted for the second and the third operand of the ?: operator to be an invocation of a void method.

The type of the expression built using ?: is determined by the types of the second and the third operands.

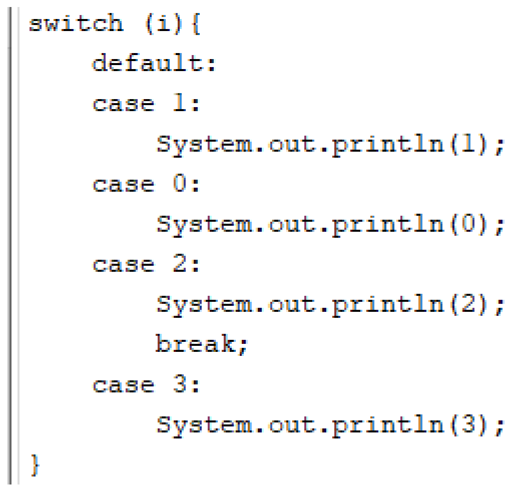
> If one of the operands is of type byte and the other is of type short, then the type of the conditional expression is short.
>
> If one of the operands is of type T where T is byte, short, or char, and the other operand is a constant expression of type int whose value is representable in type T, then the type of the conditional expression is T.

Otherwise, binary numeric promotion (5.6.2) is applied to the operand types, and the type of the conditional expression is the promoted type of the second and third operands.

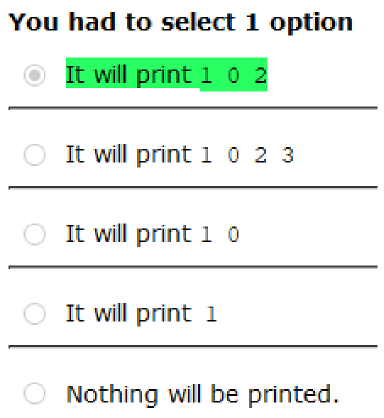
> If one of the second and third operands is of the null type and the type of the other is a reference type, then the type of the conditional expression is that reference type.
>
> If the second and third operands are of different reference types, then it must be possible to convert one of the types to the other type (call this latter type T) by assignment conversion (5.2); the type of the conditional expression is T. It is a compile-time error if neither type is assignment compatible with the other type.

Problema 11

What is the result of executing the following code when the value of i is 5:



```
switch (i){
    default:
    case 1:
        System.out.println(1);
    case 0:
        System.out.println(0);
    case 2:
        System.out.println(2);
        break;
    case 3:
        System.out.println(3);
}
```



You had to select 1 option

[✓ It will print `1 0 2`]

It will print `1 0 2 3`

It will print `1 0`

It will print `1`

Nothing will be printed.

Explanation

The type of the switch expression must be String, char, byte, short, or int (and their wrapper classes), or an enum or a compile-time error occurs.

All of the following must be true, or a compile-time error will result:

1. Every case constant expression associated with a switch statement must be assignable (5.2) to the type of the switch Expression.
2. No two of the case constant expressions associated with a switch statement may have the same value.
3. At most one default label may be associated with the same switch statement.

Basically it looks for a matching case or if no match is found it goes to default. (If default is also not found it does nothing)

Then it executes the statements till it reaches a break or end of the switch statement.

Here, it goes to default and executes till it reaches first break. So it prints 1 0 2.

Problema 12

What will the following code print?

```
  int i = 0;

  int j = 1;

  if( (i++ == 0) & (j++ == 2) ){

     i = 12;

  }

  System.out.println(i+" "+j);
```

Explanation:

The | and & operators, when applied to boolean operands, ensure that both the sides are evaluated. This is opposed to || and && operators, which do not evaluate the Right Hand Side operand if the result can be known by just evaluating the Left Hand Side.

Now, let us see the values of i and j at each step:

```
int i = 0;
int j = 1;
if( (i++ == 0) & (j++ == 2) )
/*
First, compare i with 0 and increment i, this comparison
returns true and
i becomes 1. Now Evaluate next condition:
compare j with 2 and increment j, this comparison return
false and j becomes 2.
true & false returns false so i = 12 is not executed. */
{
   i = 12;
}
System.out.println(i+" "+j)); //print 1 and 2
```

Problema 13

Which statements about the output of the following programs are true?

```
public class TestClass{
   public static void main(String args[ ] ){
      int i = 0 ;
      boolean bool1 = true;
      boolean bool2 = false;
      boolean bool  = false;
      bool = (bool2 &  method1("1"));  //1
      bool = (bool2 && method1("2"));  //2
      bool = (bool1 |  method1("3"));  //3
      bool = (bool1 || method1("4"));  //4
   }
   public static boolean method1(String str){
      System.out.println(str);
      return true;
   }
}
```

You had to select 2 options

[✓ 1 will be the part of the output.]

& (unlike &&), when used as a logical operator, does not short circuit the expression, which means it always evaluates both the operands even if the result of the whole expression can be known by just evaluating the left operand.

☐ 2 will be the part of the output.

[✓ 3 will be the part of the output.]

& and | (unlike && and ||), when used as logical operators, do not short circuit the expression, which means they always evaluate both the operands even if the result of the whole expression can be known by just evaluating the left operand.

☐ 4 will be the part of the output.

☐ None of the above

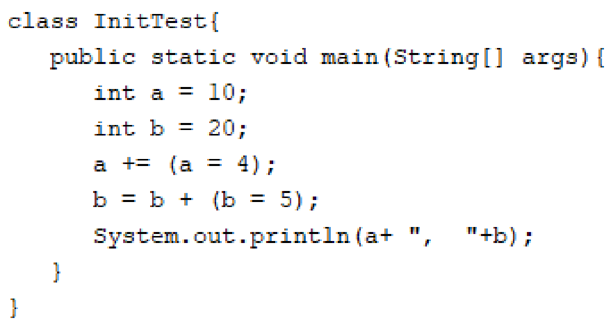
Explanation:

& and | do not short circuit the expression. The value of all the expressions (1 through 4) can be determined just by looking at the first part.

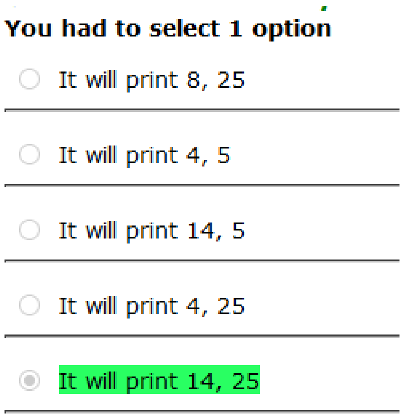
&& and || do not evaluate the rest of the expression if the result of the whole expression can be known by just evaluating the left operand, so method1() is not called for 2 and 4.

Problema 14

What will the following class print ?



```
class InitTest{
   public static void main(String[] args){
      int a = 10;
      int b = 20;
      a += (a = 4);
      b = b + (b = 5);
      System.out.println(a+ ",  "+b);
   }
}
```



You had to select 1 option

- It will print 8, 25
- It will print 4, 5
- It will print 14, 5
- It will print 4, 25
- [✓ It will print 14, 25]

Explantion:

`a += (a =4)` is same as `a = a + (a=4).`
First, a's value of 10 is kept aside and (a=4) is evaluated. The statement (a=4) assigns 4 to a and the whole statement returns the value 4. Thus, 10 and 4 are added and assigned back to a.

Same logic applies to `b = b + (b = 5);` as well.

Problema 15

What is the result of executing the following fragment of code:

```
boolean b1 = false;

boolean b2  = false;

if (b2 = b1 == false){

   System.out.println("true");

} else{

   System.out.println("false");

}
```

You had to select 1 option

- Compile time error.
- [✓ It will print `true`]
- It will print `false`
- Runtime error.
- It will print nothing.

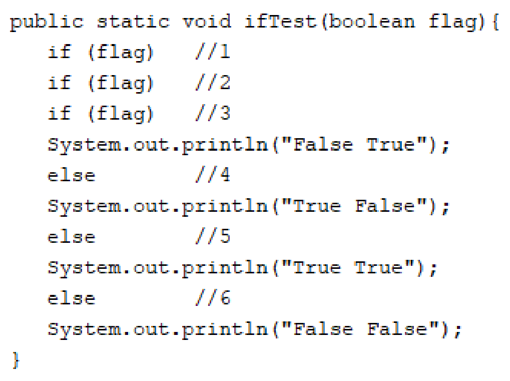
Explanation:

All that `if()` needs is a `boolean`, now `b1 == false` returns `true`, which is a `boolean` and since `b2 = true` is an expression and every expression has a return value (which is the Left Hand Side of the expression), it returns `true`, which is again a `boolean`.

FYI: the return value of expression `i = 10;` is `10` (an int).

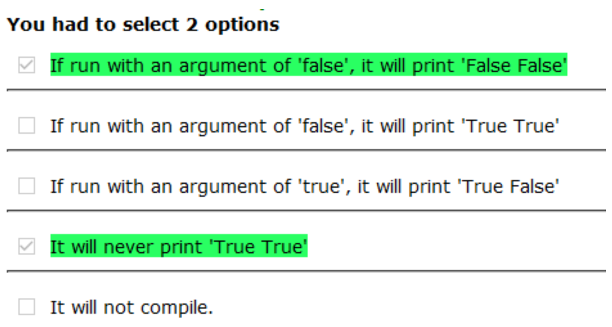
Problema 16

Consider the following method...



```
public static void ifTest(boolean flag){
   if (flag)   //1
   if (flag)   //2
   if (flag)   //3
   System.out.println("False True");
   else        //4
   System.out.println("True False");
   else        //5
   System.out.println("True True");
   else        //6
   System.out.println("False False");
}
```

Which of the following statements are correct ?



You had to select 2 options

- [✓ If run with an argument of 'false', it will print 'False False']
- If run with an argument of 'false', it will print 'True True'
- If run with an argument of 'true', it will print 'True False'
- [✓ It will never print 'True True']
- It will not compile.

Look at it like this:

```
  if (flag)      //1
   {
       if (flag)       // 2
       {
            if (flag)        //3
            {
                   System.out.println("False True");
            }
            else             //4
            {
                   System.out.println("True False");
            }
       }
       else             //5
       {
             System.out.println("True True");
       }
   }
  else             //6
  {
        System.out.println("False False");
   }
```

Note that if and else do not cascade. They are like opening an closing brackets. So, else at //4 is associated with if at //3 and else at //5 is associated with if at //2

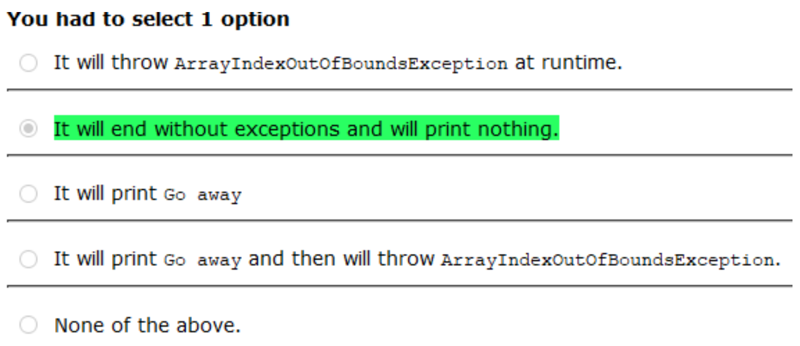
Problema 17

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.



You had to select 1 option

- It will throw `ArrayIndexOutOfBoundsException` at runtime.
- [✓ It will end without exceptions and will print nothing.]
- It will print `Go away`
- It will print `Go away` and then will throw `ArrayIndexOutOfBoundsException`.
- None of the above.

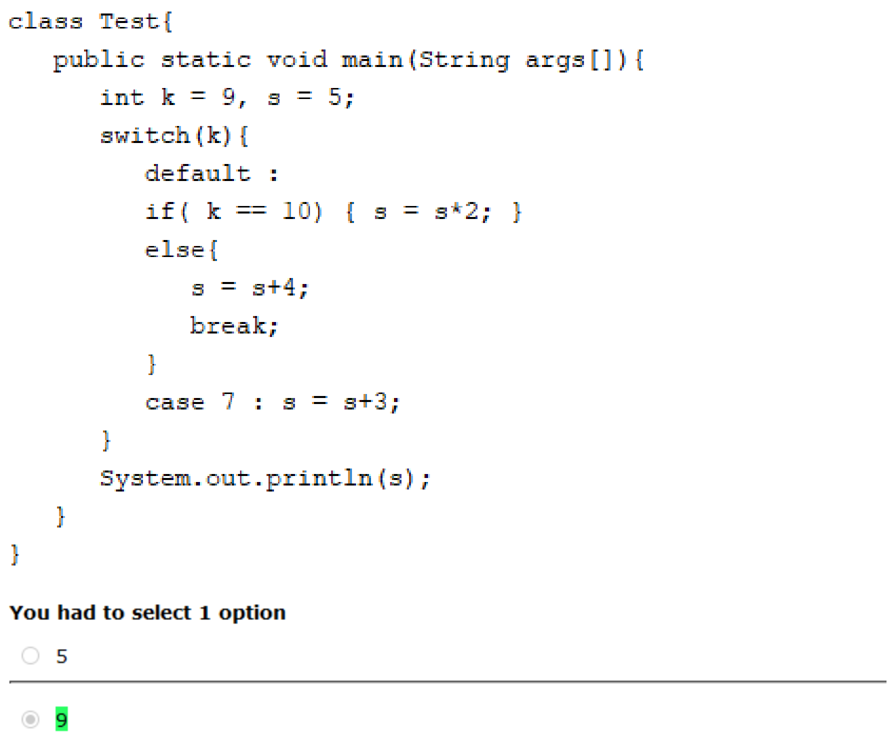
Explanation:

As in C and C++, the Java if statement suffers from the so-called "dangling else problem,"
The problem is that both the outer if statement and the inner if statement might conceivably own the else clause.
In this example, one might be tempted to assume that the programmer intended the else clause to belong to the outer if statement.

The Java language, like C and C++ and many languages before them, arbitrarily decree that an else clause belongs to the innermost if so as the first if() condition fails (args[0] not being "open") there is no else associated to execute. So, the program does nothing. The else actually is associated with the second if. So had the command line been :
`java Test open`, it would have executed the second if and thrown `ArrayIndexOutOfBoundsException`.
If the command line had been:
`java Test open xyz`, it would execute the else part(which is associated with the second if) and would have printed "`Go away xyz`".

Problema 18

What will the following program print?

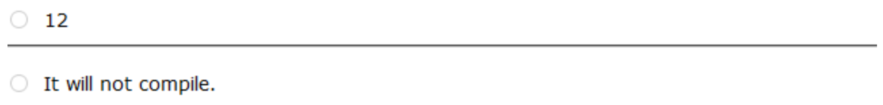


```
class Test{
   public static void main(String args[]){
      int k = 9, s = 5;
      switch(k){
         default :
         if( k == 10) { s = s*2; }
         else{
            s = s+4;
            break;
         }
         case 7 : s = s+3;
      }
      System.out.println(s);
   }
}
```

You had to select 1 option

- 5
- [✓ 9]

Since 9 does not match any of the case labels, it is accepted by default block. In this block, the else part is executed, which sets s to the value of s+4, i.e. 9. Since there is a break in the else block, case 7: is not executed.



- 12
- It will not compile.

Problem 19

Consider the following method...

```
public void ifTest(boolean flag){
   if (flag)   //1
   if (flag)   //2
   System.out.println("True False");
   else        // 3
   System.out.println("True True");
   else        // 4
   System.out.println("False False");
}
```

Which of the following statements are correct ?

You had to select 3 options

- ☑ If run with an argument of 'false', it will print 'False False'
- ☐ If run with an argument of 'false', it will print 'True True'
- ☑ If run with an argument of 'true', it will print 'True False'
- ☑ It will never print 'True True'
- ☐ It will not compile.

Explanation:

Note that if and else do not cascade. They are like opening and closing braces.

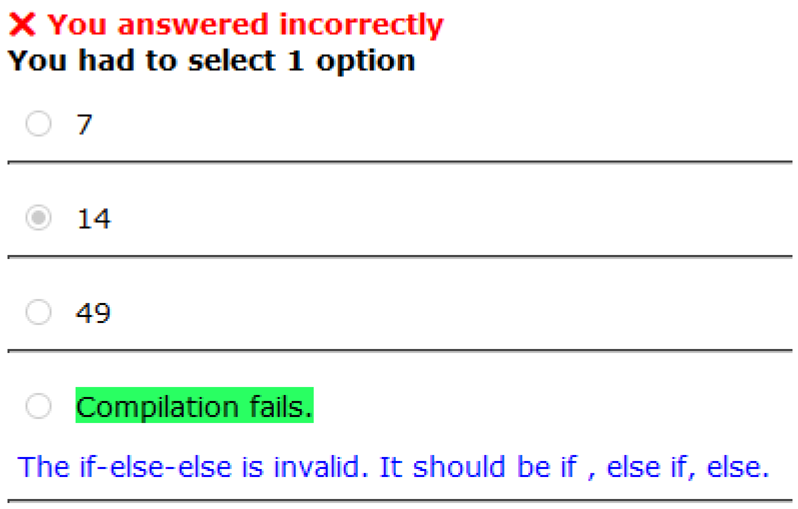
```
  if (flag)   //1
       if (flag)   //2
          System.out.println("True False");
       else        // 3 This closes //2
          System.out.println("True True");
   else        // 4 This closes //1
      System.out.println("False False");
```

So, else at //3 is associated with if at //2 and else at //4 is associated with if at //1

Problem 20

What will the following method return if called with an argument of 7?

```
public int transformNumber(int n){
    int radix = 2;
    int output = 0;
    output += radix*n;
    radix = output/radix;
    if(output<14){
        return output;
    }
    else{
        output = output*radix/2;
        return output;
    }
    else {
        return output/2;
    }
}
```



✗ You answered incorrectly

You had to select 1 option

- 7
- [✓ 14]
- 49
- Compilation fails.

The if-else-else is invalid. It should be if , else if, else.

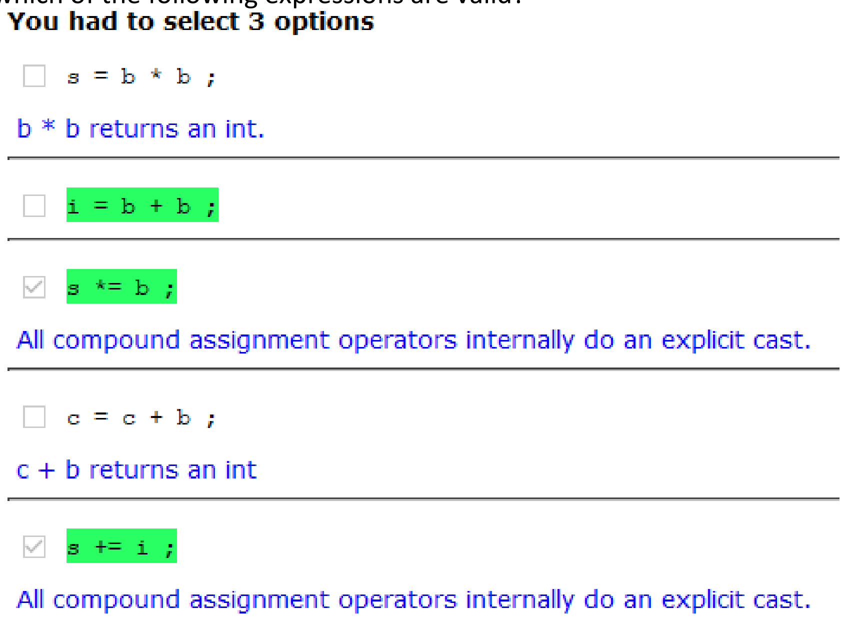
https://youtu.be/CidQTtROFq8

Problema 21

Given:

```
byte b = 1;
char c = 1;
short s = 1;
int i = 1;
```

which of the following expressions are valid?



You had to select 3 options

☐ `s = b * b ;`

b * b returns an int.

☐ `i = b + b ;`

✓ `s *= b ;`

All compound assignment operators internally do an explicit cast.

☐ `c = c + b ;`

c + b returns an int

✓ `s += i ;`

All compound assignment operators internally do an explicit cast.

Explanation:

Remember these rules for primitive types:
1. Anything bigger than an int can NEVER be assigned to an int or anything smaller than int (byte, char, or short) without explicit cast.
2. CONSTANT values up to int can be assigned (without cast) to variables of lesser size (for example, short to byte) if the value is representable by the variable.(that is, if it fits into the size of the variable).
3. operands of mathematical operators are ALWAYS promoted to AT LEAST int. (i.e. for byte * byte both bytes will be first promoted to int.) and the return value will be AT LEAST int.
4. Compound assignment operators (+=, *= etc) have strange ways so read this carefully:

A compound assignment expression of the form E1 op= E2 is equivalent to E1 = (T)((E1) op (E2)), where T is the type of E1, except that E1 is evaluated only once.
Note that the implied cast to type T may be either an identity conversion or a narrowing primitive conversion.
For example, the following code is correct:

```
short x = 3;
x += 4.6;
```

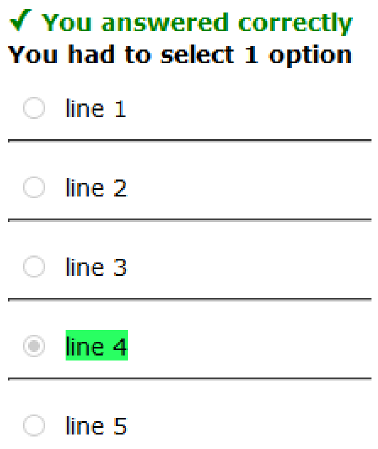
and results in x having the value 7 because it is equivalent to:

```
short x = 3;
x = (short)(x + 4.6);
```

Problema 22

Which of the lines will cause a compile time error in the following program?

```
public class MyClass{
   public static void main(String args[]){
      char c;
      int i;
      c = 'a';//1
      i = c;  //2
      i++;    //3
      c = i;  //4
      c++;    //5
   }
}
```



✓ You answered correctly

You had to select 1 option

- line 1
- line 2
- line 3
- [✓ line 4]
- line 5

1. A char value can ALWAYS be assigned to an int variable, since the int type is wider than the char type. So line 2 is valid.
2. Line 4 will not compile because it is trying to assign an int to a char. Although the value of i can be held by the char but since 'i' is not a constant but a variable, implicit narrowing will not occur.

Here is the rule given in JLS regarding assigment of constant values to primitive variables without explicit cast:

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

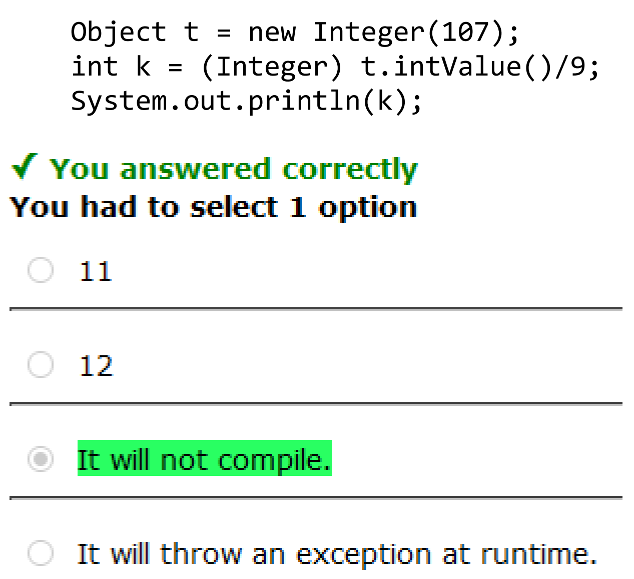
1. The expression is a compile time constant expression of type byte, char, short, or int.
2. The type of the variable is byte, short, or char.
3. The value of the expression (which is known at compile time, because it is a constant expression) is representable in the type of the variable.

Note that implict narrowing conversion (i.e. conversion without an explicit cast) does not apply to float, long, or double.

For example, char ch = 30L; will fail to compile although 30 is small enough to fit into a char.

Pregunta 23

What will the following code snippet print?



```
Object t = new Integer(107);
int k = (Integer) t.intValue()/9;
System.out.println(k);
```

✓ You answered correctly

You had to select 1 option

- 11
- 12
- [✓ It will not compile.]
- It will throw an exception at runtime.

Compiler will complain that the method intValue() is not available in Object. This is because the . operator has more precedence than the cast operator. So you have to write it like this: int k = ((Integer) t).intValue()/9;

 Now, since both the operands of / are ints, it is an integer division. This means the resulting value is truncated (and not rounded). Therefore, the above statement will print 11 and not 12.

Problema 24

What letters will be printed by this program?

```
public class ForSwitch{
    public static void main(String args[]){
        char i;
        LOOP: for (i=0;i<5;i++){
            switch(i++){
                case '0': System.out.println("A");
                case 1: System.out.println("B"); break LOOP;
                case 2: System.out.println("C"); break;
                case 3: System.out.println("D"); break;
                case 4: System.out.println("E");
                case 'E' : System.out.println("F");
            }
        }
    }
}
```

✗ You answered incorrectly
You had to select 2 options

- ☑ A
- ☑ B
- ☐ C
- ☐ D
- ☐ F

Explanation:

1. Defining i as char doesn't mean that it can only hold characters (a, b, c etc). It is an integral data type which can take any +ive integer value from 0 to 2^16 -1.
   2. Integer 0 or 1, 2 etc. is not same as char '0', '1' or '2' etc.
2.
   so when i is equal to 0, nothing gets printed and i is incremented to 1 (due to i++ in the switch).
   i is then incremented again by the for loop for next iteration. so i becomes 2.
   when i = 2, "C" is printed and i is incremented to 3 (due to i++ in the switch) and then i is incremented to 4 by the for loop so i becomes 4.
   when i = 4, "E" is printed and since there is no break, it falls through to case 'E' and "F" is printed.
   i is incremented to 5 (due to i++ in the switch) and then it is again incremented to 6 by the for loop. Since i < 5 is now false, the for loop ends.

https://youtu.be/-5EEtUpOUTI