01 Java Working with Data Types

Problema 1

Note: This question may be considered too advanced for this exam.

Which statements can be inserted at line 1 in the following code to make the program write x on the standard output when run?

```
public class AccessTest{
   String a = "x";
   static char b = 'x';
   String  c = "x";
   class Inner{
      String  a = "y";
      String  get(){
         String c = "temp";
         // Line 1
         return c;
      }
   }
   AccessTest() {
     System.out.println(  new Inner().get()  );
   }
   public static void main(String args[]) {  new AccessTest();  }
}
```

You had to select 3 options

- [x] `c = c;`

It will reassign 'temp' to c!

- [x] `c = this.a;`

It will assign "y" to c.

- [x] `c = ""+AccessTest.b;`

Because b is static.

- [ ] `c = AccessTest.this.a;`

- [ ] `c = ""+b;`

Problema 2

Which of the following are valid at line 1?

```
public class X{

    //line 1: insert code here.

}
```

✓ **You answered correctly**
You had to select 2 options

☑ String s;

☐ String s = 'asdf';

A string must be enclosed in double quotes ".

☐ String s = 'a';

'a' is a char. "a" is a String.

☑ String s = this.toString();

Since every class directly or indirectly extends Object class and since Object class has a toString() method, that toString() method will be invoked and the String that it returns will be assigned to s.

☐ String s = asdf;

there is no variable asdf defined in the given class.

Problema 3

What will the following code print?

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

        int x = 1____3;   //1
        long y = 1_3;     //2
        float z = 3.234_567f; //3

        System.out.println(x+" "+y+" "+z);
    }
}
```

You had to select 1 option

- Compilation error at //1
- Compilation error at //2
- Compilation error at //3
- Compilation error at //1 and //3
- 10003 103 3.234567
- 13 13 3.234567

The number at //1 and //2 are actually the same. Although confusing, it is legal to have multiple underscores between two digits.

Explanation:

You may use underscore for all kinds of numbers including long, double, float, binary, as well as hex. For example, the following are all valid numbers - int hex = 0xCAFE_BABE; float f = 9898_7878.333_333f; int bin = 0b1111_0000_1100_1100;

Problema 4

Given:

```
String mStr = "123";

long m = // 1
```

Which of the following options when put at //1 will assign 123 to m?

You had to select 3 options

- [x] `new Long(mStr);`

  Auto unboxing will occur.

- [x] `Long.parseLong(mStr);`

- [ ] `Long.longValue(mStr);`

  longValue is a non-static method in Long class.

- [ ] `(new Long()).parseLong(mStr);`

  Long (or any wrapper class) does not have a no-args constructor, so new Long() is invalid.

- [x] `Long.valueOf(mStr).longValue();`

  `Long.valueOf(mStr)` returns a Long object containing 123. `longValue()` on the Long object returns 123.

Problema 5

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

```
public class TestClass{
   public static void main(String args[ ] ){
      int i, j, k;
      i = j = k = 9;
      System.out.println(i);
   }
}
```

- ☐ The code will not compile as 'j' is being used before getting initialized.

j is being initialize by the expression k = 9, which evaluates to 9.

- ☑ The code will compile correctly and will display '9' when run.

- ☐ The code will not compile as 'j' and 'i' are being used before getting initialized.

- ☑ All the variables will get a value of 9.

Realmente se puede pasar el valor de varias en la misma línea. 'chained' mencionan que se le dice y que es posible como en C.

= can be chained. For example, assuming all the variables are declared appropriately before hand, a = b = c = d; is valid. However, chaining to use a value of a variable at the time of declaration is not allowed. For example, int a = b = c = 100; is invalid if b and c are not already declared. Had b and c been already declared, int a = b = c = 100; would have been valid.

Another implication of this is :

boolean b = false;

if(b = true) { System.out.println("TRUE");}

The above code is valid and will print TRUE. Because b = true has a boolean value, which is what an if statement expects.

Note that if(i = 5) { ... } is not valid because the value of the expression i = 5 is an int (5) and not a boolean.

Problema 6

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));
    }
}
```

https://youtu.be/otbcOmaq1vk

Problema 7

Which of the following are correct ways to initialize the static variables MAX and CLASS_GUID ?

```
class Widget{
   static int MAX;     //1
   static final String CLASS_GUID;   // 2
   Widget(){
       //3
   }
   Widget(int k){
       //4
   }
}
```

You had to select 2 options

- [x] Modify lines //1 and //2 as : `static int MAX = 111; static final String CLASS_GUID = "XYZ123";`

You can initialize both the variables at declaration itself.

- [ ] Add the following line just after //2 : `static {  MAX = 111; CLASS_GUID = "XYZ123"; }`

Initializing the static variables in a static block ensures that they are initialized even when no instance of the class is created.

- [x] Add the following line just before //1 : `{ MAX = 111; CLASS_GUID = "XYZ123"; }`

This is not a static initializer and so will not be executed until an instance is created.

- [ ] Add the following line at //3 as well as //4 : `MAX = 111; CLASS_GUID = "XYZ123";`

This works for non-static final fields but not for static final fields.

- [ ] Only option 3 is valid.

The rules are:

1. static variables can be left without being explicitly initialized. (They will get default values).

2. final variables must be explicitly initialized.

Now, here CLASS_GUID is a 'static final' variable and not just final or static. As static fields can be accessed even without creating an instance of the class, it is entirely possible that this field can be accessed before even a single instance is created. In this case, no constructor or non-static initializer had ever been called. And so, the field (as it is final and so must be initialized explicitly) remains uninitialized. This causes the compiler to complain

https://youtu.be/AaMlTr4wy08

Problema 8

In the following code, after which statement (earliest), the object originally held in s, may be garbage collected ?

```
1. public class TestClass{
2.   public static void main (String args[]){
3.      Student s = new Student("Vaishali", "930012");
4.      s.grade();
5.      System.out.println(s.getName());
6.      s = null;
7.      s = new Student("Vaishali", "930012");
8.      s.grade();
9.      System.out.println(s.getName());
10      s = null;
     }
   }
public class Student{
   private String name, rollNumber;
   public Student(String name, String rollNumber) {
      this.name = name;
      this.rollNumber = rollNumber;
   }
   //valid setter and getter for name and rollNumber follow
   public void grade() {
   }
}
```

- Line 6

Explanation:

In this case, since there is only one reference to Student object, as soon as it is set to null, the object held by the reference is eligible for GC, here it is done at line 6. Note that although an object is created at line 7 with same parameters, it is a different object and it will be eligible for GC after line 10.

Problema 9

Assume that a, b, and c refer to instances of primitive wrapper classes. Which of the following statements are correct?

You had to select 2 options

- [ ] a.equals(a) will always return true.
- [x] b.equals(c) may return false even if c.equals(b) returns true.
- [ ] a.equals(b) returns same as a == b.

The wrapper classes's equals() method overrides Object's equals() method to compare the actual value instead of the reference.

- [ ] a.equals(b) throws an exception if they refer to instances of different classes.

It returns false in such a case.

- [x] a.equals(b) returns false if they refer to instances of different classes.

Para entender por qué las respuestas sirve la función, si son de diferente clase.

Equals method of a primitive wrapper class (e.g. java.lang.Integer, Double, Float etc) are

1. symmetric => a.equals(b) returns same as b.equals(a)
2. transitive => if a.equals(b) and b.equals(c) return true, then a.equals(c) returns true.
3. reflexive => a.equals(a) return true.

For example, the following code for the equals method on Integer explains how it works:

```
public boolean equals(Object obj) {
    if (obj instanceof Integer) {
        return value == ((Integer)obj).intValue();
    }
    return false;
 }
```

Problema 10

Given that TestClass is a class, how many objects and reference variables are created by the following code?

```
TestClass t1, t2, t3, t4;

t1 = t2 = new TestClass();

t3 = new TestClass();
```

You had to select 1 option

- ○ 2 objects, 3 references.
- ◉ 2 objects, 4 references.

two news => two objects. t1, t2, t3, t4 => 4 references.

- ○ 3 objects, 2 references.
- ○ 2 objects, 2 references.
- ○ None of the above.

Explanation:

A declared reference variable exists regardless of whether a reference value (i.e. an object) has been assigned to it or not.

Problema 11

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?

You had to select 2 options

☑ `al.add(111);`

You cannot box an int into a Double object.

☐ `System.out.println(al.indexOf(1.0));`

`indexOf`'s accepts Object as a parameter. Although 1.0 is not an object, it will be boxed into a Double object.

☐ `System.out.println(al.contains("string"));`

☑ `Double d = al.get(al.length);`

ArrayList does not have a field named length. It does have a method named size() though. So you can do: Double d = al.get(al.size()); It will compile but will throw IndexOutOfBoundsException at run time in this case because al.size() will return 0 and al.get(0) will try to get the first element in the list.

Explanation:

Note that al is declared as ArrayList<Double>, therefore the add method is typed to accept only a Double.

- La primera respuesta puede ser sin punto decimal y mandará como resultado un '-1'
- Por parte de la segunta es como si buscara dentro del ArrayList el string "string" y si no lo encuentra da un valor de 'false'.

https://youtu.be/nbKIzNGtaeY

Problema 12

After which line will the object created at line XXX be eligible for garbage collection?

```
public Object getObject(Object a) //0
   {


Object b = new Object();  //XXX


Object c, d = new Object(); //1
c = b; //2
b = a = null; //3
return c; //4
}
```

You had to select 1 option

- ○ //2
- ◉ //3
- ○ //4
- ○ Never in this method.
- ○ Cannot be determined.

Ponte chingon Papi, la 'c' sigue señalando al objeto creado en 'b'

Explanation:

Note that at line 2, c is assigned the reference of b. i.e. c starts pointing to the object created at //XXX. So even if at //3 b and a are set to null, the object is not without any reference. Also, at //4 c is being returned. So the object referred to by c cannot be garbage collected in this method!

Problema 13

When is the Object created at line //1 eligible for garbage collection?

```
public class TestClass{

  public Object getObject(){

     Object obj = new String("aaaaa");   //1

     Object objArr[] = new Object[1]; //2

     objArr[0] = obj; //3

     obj = null;      //4

     objArr[0] = null;//5

     return obj;      //6

  }

}
```

You had to select 1 option

- ○ Just after line 2.
- ○ Just after line 3.
- ◉ Just after line 4.
- ○ Just after line 5.
- ○ Just after line 6.

The official exam objectives now explicitly mention Garbage collection. All you need to know is:

1. An object can be made eligible for garbage collection by making sure there are no references pointing to that object.
2. You cannot directly invoke the garbage collector. You can suggest the JVM to perform garbage collection by calling System.gc();

- After line 3, both, obj and objArr[0] are pointing to the same String object.
- After line 4, obj points to null but objArr[0] is still pointing to the String object.
- After line 5, objArr[0] also starts pointing to null so there is no reference left that is pointing to the String object. So it is now available for Garbage collection.

https://youtu.be/7pbtNf_loRs

Problema 14

The wollowing code snippet will print 4

```
int i1 = 1, i2 = 2, i3 = 3;
int i4 = i1 + (i2=i3 );
System.out.println(i4);
```

✓ You answered correctly
You had to select 1 option

- (•) True
- ( ) False

First the value of i1 is evaluated (i.e. 1). Now, i2 is assigned the value of i3 i.e. i2 becomes 3. Finally i4 gets 1 +3 i.e. 4.

I2 apunta al valor de i3

I2 = 3 y es válido en Java para asignarlo dentro del paréntesis por plo que solo se suma el valor 3

El resultado es la suma 1 + 3 = 4.

Problema 15

Which of the changes given in options can be done (independent of each other) to let the following code compile and run without errors when its generateReport method is called?

```
class SomeClass{
   String s1 = "green mile";   // 0
   public void generateReport( int n ){
      String local;   // 1
      if( n > 0 ) local = "good";   //2
      System.out.println( s1+" = " + local );   //3
   }
}
```

You had to select 2 options

- [ ] Insert after line 2 : `else local = "bad";`
- [ ] Insert after line 2 : `if(n <= 0) local = "bad";`
- [x] Move line 1 and place it after line 0.
- [x] change line 1 to : `final String local = "rocky";`

Making it final will not let //2 compile as it would then try to modify a final variable.

- [ ] The program already is without any errors.

Explanation:

The problem is that local is declared inside a method is therefore local to that method. It is called a local variable (also known as automatic variable) and it cannot be used before initialized. Further, it will not be initialized unless you initialize it explicitly because local variables are not initialized by the JVM on its own. The compiler spots the usage of such uninitialized variables and ends with an error message.

1. Making it a member variable (choice "Move line 1 and place it after line 0.") will initialize it to null.
2. Putting an else part (choice "Insert after line 2 : else local = "bad";") will ensure that it is initialized to either 'good' or 'bad'. So this also works.
Choice "Insert after line 2 : if(n <= 0) local = "bad";" doesn't work because the second 'if' will actually be another statement and is not considered as a part of first 'if'. So, compiler doesn't realize that 'local' will be initialized even though it does get initialized.

Problema 16

Consider the following code:

```
class MyClass { }

public class TestClass{

   MyClass getMyClassObject(){

      MyClass mc = new MyClass(); //1

         return mc; //2

   }

   public static void main(String[] args){

      TestClass tc = new TestClass(); //3

      MyClass x = tc.getMyClassObject(); //4

      System.out.println("got myclass object"); //5

      x = new MyClass(); //6

      System.out.println("done"); //7

   }

}
```

After what line the MyClass object created at line 1 will be eligible for garbage collection?

✓ **You answered correctly**
You had to select 1 option

- ○ 2
- ○ 5
- ◉ 6

At line 6, x starts pointing to a new MyClassObject and no reference to the original MyClass object is left.

- ○ 7
- ○ Never till the program ends.

The official exam objectives now explicitly mention Garbage collection. All you need to know is:

1. An object can be made eligible for garbage collection by making sure there are no references pointing to that object.
2. You cannot directly invoke the garbage collector. You can suggest the JVM to perform garbage collection by calling System.gc();

https://youtu.be/8KhChKzBmjM

Problema 17

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

```
public class TestClass{
  public static void main(String args[ ] ){
    Object a, b, c ;
    a = new String("A");
    b = new String("B");
    c = a;
    a = b;
    System.out.println(""+c);
  }
}
```

You had to select 1 option

- The program will print `java.lang.String@XXX`, where XXX is the memory location of the object a.
- The program will print A
- The program will print B
- The program will not compile because the type of a, b, and c is Object.

String is an Object as well. You can always assign an object of the subclass to a super class reference without a cast.

- The program will print `java.lang.String@XXX`, where XXX is the hash code of the object a.

Explanation

The variables a, b and c contain references to actual objects. Assigning to a reference only changes the reference value, and not the object pointed to by the reference. So, when c = a is executed c starts pointing to the string object containing A. and when a = b is executed, a starts pointing to the string object containing B but the object containing A still remains same and c still points to it. So the program prints A and not B.

The Object class's toString generates a string using: getClass().getName() + '@' + Integer.toHexString(hashCode())
But in this case, String class overrides the toString() method that returns just the actual string value.

Problema 18

What will the following code print when run?

```
public class TestClass{
    public static Integer wiggler(Integer x){
       Integer y = x + 10;
       x++;
       System.out.println(x);
       return y;
    }
    public static void main(String[] args){
       Integer dataWrapper = new Integer(5);
       Integer value = wiggler(dataWrapper);
       System.out.println(dataWrapper+value);
    }
}
```

You had to select 1 option

- 5 and 20
- 6 and 515
- 6 and 20
- 6 and 615
- It will not compile.

1. Wrapper objects are always immutable. Therefore, when dataWrapper is passed into wiggler() method, it is never changed even when x++; is executed. However, x, which was pointing to the same object as dataWrapper, is assigned a new Integer object (different from dataWrapper) containing 6.

2. If both the operands of the + operator are numeric, it adds the two operands. Here, the two operands are Integer 5 and Integer 15, so it unboxes them, adds them, and prints 20.

You may want to check out the free video by Dr. Sean Kennedy explaining this question: https://youtu.be/5Yk2-Rmkh0M

Problema 19

Which is the earliest line in the following code after which the object created on line // 1 can be garbage collected, assuming no compiler optimizations are done?

```
public class NewClass{
   private Object o;
   void doSomething(Object s){  o = s;   }

   public static void main(String args[]){
      Object obj = new Object(); // 1
      NewClass tc = new NewClass(); //2
      tc.doSomething(obj); //3
      obj = new Object();    //4
      obj = null;    //5
      tc.doSomething(obj); //6
   }
}
```

You had to select 1 option

- ○ Line 1
- ○ Line 2
- ○ Line 3
- ○ Line 4
- ◉ Line 5
- ○ Line 6

Before this line the object is being pointed to by at least one variable.

Hasta ese punto señalará por medio del método doSomething que obj apunte a null.

Problema 20

Which of the following options will yield a Boolean wrapper object containing the value true?

You have to select 3 options:

- `Boolean.parseBoolean(" true ")`

// This will return false because of the extra spaces at the ends. Remember that case of the argment is ignored but spaces are not.

- `Boolean.parseBoolean("true")`

// Although this will return true but it is still not a valid answer because parseBoolean returns a primitive and not a Boolean wrapper object.

- [x] `Boolean.valueOf(true)`
- [ ] `Boolean.valueOf("trUE")`
- [x] `Boolean.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.

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.

Problema 21

What will the following program print?

```
public class TestClass{
  public static void main(String[] args){
    unsigned byte b = 0;
    b--;
    System.out.println(b);
  }
}
```

You had to select 1 option

- 0
- -1
- 255
- -128
- It will not compile.

There no unsigned keyword in java! A char can be used as an unsigned integer.

Problema 22

Which of the following are valid code snippets appearing in a method:

- [ ] `int a = b = c = 100;`

Chaining to use a value of a variable at the time of declaration is not allowed. Had b and c been already declared, it would have been valid. For example, the following is valid:

```
int  b = 0, c = 0;

int a = b = c = 100;
```

Even the following is valid:

```
int  b , c;  //Not initializing b and c here.

int a = b = c = 100; //declaring a and initializing c, b, and a at the same time.
```

Notice the order of initialization of the variables - c is initialized first, b is initialized next by assigning to it the value of c. Finally, a is initialized.

- [x] `int a, b, c; a = b = c = 100;`

- [x] `int a, b, c=100;`

- [x] `int a=100, b, c;`

- [ ] `int a= 100 = b = c;`

Explanation:

Java does not allow chained initialization in declaration so option 1 and 5 are not valid.

Ver Problema 5

Problema 23

Identify the valid code fragments when occurring by themselves within a method.

You had to select 1 option

- `long y = 123_456_L;`

An underscore can only occur in between two digits. So the _ before L is invalid.

- `long z = _123_456L;`

An underscore can only occur in between two digits. So the _ before 1 is invalid.
_123_456L is a valid variable name though. So the following code is valid:

```
int _123_456L = 10;
long z = _123_456L;
```

An exception to this rule is that multiple continuous underscores can appear between two digits.

For example, 2____3 is as good as 2_3 and is same as 23.

- `float f1 = 123_.345_667F;`

An underscore can only occur in between two digits. So the _ before . is invalid.

- `float f2 = 123_345_667F;`

- None of the above declarations are valid.

You may use underscore for all kinds of numbers including long, double, float, binary, as well as hex. For example, the following are all valid numbers -

```
int hex = 0xCAFE_BABE;
float f = 9898_7878.333_333f;
int bin = 0b1111_0000_1100_1100;
```

**No se puede usar al inicio, al final o justo a lado de un 'period'

Problema 24

Consider the following lines of code:

```
Integer i = new Integer(42);

Long ln = new Long(42);

Double d = new Double(42.0);
```

Which of the following options are valid code fragments?

You had to select 3 options

- [ ] `i == ln;`

This will fail at compile time

- [ ] `ln == d;`

This will fail at compile time

- [x] `i.equals(d);`

- [x] `d.equals(ln);`

- [x] `ln.equals(42);`

Due to auto-boxing int 42 is converted into an Integer object containing 42. So this is valid. It will return false though because ln is a Long and 42 is boxed into an Integer.

Explanation:

The concept to understand here is as follows - If the compiler can figure out that something can NEVER happen, then it flags an error. In this question, the compiler knows that ln, i or d can never point to the same object in any case because they are references to different classes of objects that have no relation (superclass/subclass) between themselves.

Nunca serán el mismo valor en memoria las primeras 2 opciones.

Problema 25

What will the following code print?

```
public class TestClass{
  static char ch;
  static float f;
  static boolean bool;

  public static void main(String[] args){
    System.out.print(f);
    System.out.print(" ");
    System.out.print(ch);
    System.out.print(" ");
    System.out.print(bool);
}
}
```

You had to select 1 option

- (●) 0.0 false
- ( ) 0.0f false
- ( ) 0.0 0 false
- ( ) 0.0 true
- ( ) 0.0f true
- ( ) 0.0f 0 true

Explanation:

This question tests you on two aspects –

1. the default values that are given to variables of various primitive types. You should remember that all numeric types, including char, get the value of 0 (or 0.0 for float and double) and boolean gets the value of false.

2. how the value is printed by System.out.print method - java.lang.System class has a public static variable named out, which is of class java.io.PrintStream. The PrintStream class has multiple

print/println methods for printing out primitive values as well as objects. For byte, short, and int, these print/println methods simply print the integer value as it is.

For char, the print/println methods translate the character into one or more bytes according to the platform's default character encoding. That is why while printing a char value of 0, a blank space is printed instead of 0 (even though the char's integral value is 0).

For long, float, and double values, these print/println methods use the respective primitive wrapper class's toString method to get the String representation of that primitive. For example, to print the value of a float variable f, it internally calls Float.toString(f). Now, this method doesn't append an "f" at the end of a float value. That is why a float value of 0.0 is printed as 0.0 and not 0.0f.

Problema 26

Given:

```
public class TestClass{
     public static void main(String[] args){
     int i = Integer.parseInt(args[1]);  // pasa el valor 2 a i
     System.out.println(args[i]);  // seria un args[2], no existe
   }
}
```

What will happen when you compile and run the above program using the following command line:

```
java TestClass 1 2
```

You had to select 1 option

- ○ It will print `1`
- ◉ It will print `2`
- ○ It will print some junk value.
- ○ It will throw `ArrayIndexOutOfBoundsException`.
- ○ It will throw `NumberFormatException`

Note: NumberFormatException extends IllegalArgumentException, which in turn extends RuntimeException.

Explanation:

Era sencilla, solo se salía del rango del Array.

Problema 27

Which of the following statements will print true when executed?

Aquí lo importante o diferencia del otro es que no indican que sean un Boolean Wrapper, sino que den de resultado un 'true'

You had to select 3 options

- [x] `System.out.println(Boolean.parseBoolean("true"));`

- [ ] `System.out.println(new Boolean(null));`

This will print false.

- [x] `System.out.println(new Boolean());`

This will not compile because Boolean class does not have a no-args constructor. Remember that no other wrapper class has a no-args constructor either. So new Integer(), or new Long() will also not compile.

- [x] `System.out.println(new Boolean("true"));`

Case of the String parameter does not matter. As long as the String equals "true" after ignoring the case, it will be parsed as true. However, if you have extra spaces, for example, " true" or "true ", it will be parsed as false.

- [ ] `System.out.println(new Boolean("trUE"));`

C ase of the String parameter does not matter. As long as the String equals "true" after ignoring the case, it will be parsed as true. However, if you have extra spaces, for example, " true" or "true ", it will be parsed as false.

Explanation:

Ver Problema 20

Problema 28

Identify correct statement(s) about the following code:

```
int value = 1,000,000; //1
switch(value){
    case 1_000_000 : System.out.println("A million 1"); //2
        break;
    case 1000000 : System.out.println("A million 2"); //3
        break;
}
```

You had to select 1 option

- ○ It will print `A million 1` when compiled and run.
- ○ It will print `A million 2` when compiled and run.
- ◉ Compilation fails only at line //1
- ○ Compilation fails only at line //2
- ○ Compilation fails only at line //3
- ○ Compilation fails at line //1 and //3

Explanation:

1. You may use underscores (but not commas) to format a number for better code readability. So //1 is invalid.

2. Adding underscores doesn't actually change the number. The compiler ignores the underscores. So 1_000_000 and 1000000 are actually same and you cannot have two case blocks with the same value. Therefore, the second case at //3 is invalid.

Problema 29

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

```
public class InitClass{
   public static void main(String args[ ] ){
      InitClass obj = new InitClass(5);
   }
   int m;
   static int i1 = 5;
   static int i2 ;
   int  j = 100;
   int x;
   public InitClass(int m){
      System.out.println(i1 + "  " + i2 + "   " + x + "  " + j + "  "+m);
   }
  { j = 30; i2 = 40; }  // Instance Initializer
   static { i1++; }      // Static Initializer
}
```

You had to select 1 option

- ○ The code will fail to compile since the instance initializer tries to assign a value to a static member.
- ○ The code will fail to compile since the member variable x will be uninitialized when it is used.
- ◉ The code will compile without error and will print 6 40 0 30 5 when run.
- ○ The code will compile without error and will print 5, 0, 0, 100, 5 when run.
- ○ The code will compile without error and will print 5, 40, 0, 30, 0 when run.

Explanation:

The value 5 is passed to the constructor to the local (automatic) variable m. So the instance variable m is shadowed. Before the body of the constructor is executed, the instance initializer is executed and assigns values 30 and 40 to variables j and i2, respectively. A class is loaded when it is first used. For example,

```
class A1{
  static int i = 10;
  static { System.out.println("A1 Loaded "); }
}
public class A{
  static { System.out.println("A Loaded "); }
  public static void main(String[] args){
    System.out.println(" A should have been loaded");
    A1 a1 = null;
    System.out.println(" A1 should not have been loaded");
    System.out.println(a1.i);
  }
}
```

When you run it you get the output:

```
A Loaded
 A should have been loaded
 A1 should not have been loaded
A1 Loaded
10
```

Problema 30

Given the following class, which statements can be inserted at line 1 without causing the code to fail compilation?

```
public class TestClass{
   int a;
   int b = 0;
   static int c;
   public void m(){
      int d;
      int e = 0;
      // Line 1
   }
}
```

You had to select 4 options

☑ a++;

Here, 'a' is an instance variable of type int. Therefore, it will be given a default value of Zero and so it need not be initialized explicitly.

☑ b++;

☐ c++;

Here 'c' is a class variable (also called as static variable) of type int so it will be given a default value of Zero and so it need not be initialized explicitly.

☑ d++;

This will not compile because 'd' is not initialized. Note that automatic variables (also called as method local variables i.e. variables declared within a method) initialized.

☑ e++;

---

d++ has to be explicitly initializated.

Problema 31

Which of these assignments are valid?

You had to select 3 options

- [x] `short s = 12 ;`

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

- [x] `long g = 012 ;`

012 is a valid octal number.

- [ ] `int i = (int) false;`

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

- [x] `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.

Problema 32

Which of the following are valid classes?

You had to select 1 option

```
public class ImaginaryNumber extends Number {
//implementation for abstract methods of the base class
}
```

`Number` is not a final class so you can extend it.

```
public class ThreeWayBoolean extends Boolean {
//implementation for abstract methods of the base class
}
```

```
public class NewSystem extends System {
//implementation for abstract methods of the base class
}
```

```
public class ReverseString extends String {
//implementation for abstract methods of the base class
}
```

Explanation

`String`, `StringBuilder`, and `StringBuffer` - all are final classes.

1. Remember that wrapper classes for primitives (`java.lang.Boolean`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Short` etc.) are also final and so they cannot be extended.

2. `java.lang.Number`, however, is not final. `Integer`, `Long`, `Double` etc. extend Number.

3. `java.lang.System` is final as well.