00 Java Basics

Problema 1

Consider the following class written by a novice programmer.

```
class Elliptical{
    public int radiusA, radiusB;
    public int sum = 100;

    public void setRadius(int r){
        if(r>99) throw new IllegalArgumentException();
        radiusA = r;
        radiusB = sum - radiusA;
    }
}
```

After some time, the requirements changed and the programmer now wants to make sure that radiusB is always (200 - radiusA) instead of (100 - radiusA) without breaking existing code that other people have written. Which of the following will accomplish his goal?

- Make sum = 200;
- Make sum = 200 and make it private.
- Make sum = 200 and make all fields (radiusA, radiusB, and sum) private.
- Write another method setRadius2(int r) and set radiusB accordingly in this method.
- His goal cannot be accomplished.
- This class will not compile.

Explanation:

setRadius method makes sure that radiusB is set to sum - radiusA. So changing sum to 200 should do it. However, note that radiusA, radiusB, and sum are public which means that any other class can access these fields directly without going through the setRadius method. So there is no way to make sure that the value of radiusB is correctly set at all times. If you make them private now, other classes that are accessing the fields directly will break. The class should have been coded with proper encapsulation of the fields in the first place.

https://youtu.be/EF6rckZHcSA

Problema 2

The following are the complete contents of TestClass.java file. Which packages are automatically imported?

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

- java.util
  - // Util tiene que ser importado
- system
  - // System is not a package. It is a class in java.lang package.
- java.lang
  - // Se incorpora automáticamente, vi que desde el IDE dice que viene cuando gneras un proyecto
- java.io
  - // No lo hace automáticamente
- String
  - // String is a class in java.lang package.
- The package has no name
  - // Todas las demás son error, por lo que, si o si esta es la respuesta, aunque no estes seguro, además,
  - // Si no pones una clase en un paquete, entonces esa clase se coloca en ese paquete sin nombre
  - // Además no podrás importar el paquete porque no tiene nombre

If there is no package statement in the source file, the class is assumed to be created in an unnamed package that has no name. In this case, all the types created in this package will be available to this class without any import statement.

However, note that this unnamed package cannot be imported in classes that belong to any other package at all, not even with any import statement. So for example, if you have a class named SomeClass in package test, you cannot access TestClass defined in the problem statement (as it belongs to the unnamed package) at all because there is no way to import it.

As per JLS Section 7.5: A type in an unnamed package has no canonical name, so the requirement for a canonical name in every kind of import declaration implies that (a) types in an unnamed package cannot be imported, and (b) static members of types in an unnamed package cannot be imported. https://youtu.be/djGcyqxyCA4

Problem 3

Which of the following are valid declarations of the standard main method?

Please Select 2 options:

a) static void main(String args[ ]) { }
   a. // Although practically correct but for the purpose of this exam you should not select this option because the method is not declared public.
b) public static int main(String args[ ]) {}
   a. // This method returns an 'int' instead of 'void'.
c) public static void main (String args) { }
   a. // The method takes only one String instead of String[].
d) final static public void main (String[ ] arguments) { }
e) public static void main (String[ ] args) { }

Explanation:

A valid declaration of "the" main() method must be public and static, should return void, and should take a single array of Strings as a parameter.

The order of the static and public keywords is irrelevant. But the return type should always come just before the method name.

Applying final to the method does not change the method signature.

In some versions of JDK, even a private or protected main() method works from command line. However, for the purpose of Java Certification exam, it should be assumed that for the JVM to execute a class using the standard main method, the accessibility of the main method must be public.

Problema 4

Which of the following are correct about "encapsulation"?

Please Select 2 options:

- Encapsulation is same as polymorphism.
  - // Es no ingresar a ciertos atributos y métodos al abstraerlos
- It helps make sure that clients have no accidental dependence on the choice of representation
  - // SI no estamos mal, es la manera en que los clientes se acostumbrer a usar métodos de ngreso y no cambiar directamente la variable
- It helps avoiding name clashes as internal variables are not visible outside.
  - // Regresando, no puede haber interferencia como introducir otro tipo de valor que ya no esté disponible en una actualización
- Encapsulation makes sure that messages are sent to the right object at run time.
  - // This is dynamic binding, an outcome of polymorphism.
- Encapsulation helps you inherit the properties of another class.

Explanation:

Encapsulation is the technique used to package the information in such a way as to hide what should be hidden, and make visible what is intended to be visible. In simple terms, encapsulation generally means making the data variables private and providing public accessors.

https://youtu.be/l9UE4NkIBVI

Problema 5

What will the following code print when compiled and run?

```
public class ATest {

    String global = "111";  // Variable de instancia de clase

    public int parse(String arg){

        int value = 0;

        try{

            String global = arg;     // variable local con el mismo nombre, es posible

            value = Integer.parseInt(global);    // Referencia a la variable local '333'

        }

        catch(Exception e){

            System.out.println(e.getClass());

        }

        System.out.print(global+" "+value+" ");

        return value;

    }

       public static void main(String[] args) {

          ATest ct = new ATest(); // Genera desde 'Object' con su constructor

           System.out.print(ct.parse("333"));

       }

}
```

- 111 333 333
- 333 333 333
- java.lang.NumberFormatException
- java.lang.Exception
- Compilation fails

Explanation: https://youtu.be/nTNpyfrECiU

Observe that a new local variable named global is defined within a try block. It is accessible only within the try block. It also shadows the instance field of the same name global within the try block. It is this variable that is used in parseInt. Therefore, value is set to 333.
However, when you print global in parse method, the global defined in the try block is out of scope and the instance field named global is used. Therefore, it prints 111.

```
// El resultado imprimirá 111 local, 333 value y 333 return sysout
```

There is no exception because 333 can be parsed into an int. If you pass a string that cannot be parsed into an int to the parseInt method, it will throw a java.lang.NumberFormatException.

Problema 6

Given the following class, which of these are valid ways of referring to the class from outside of the package com.enthu?

```
package com.enthu;
public class Base{
   // lot of code...
}
```

Please select 2 options:

- Base
  - // Only if you import the whole package containing the class or import the class first.
- By importing the package com.* and referring to the class as enthu.Base
  - // package 'com' does not contain Base.
- importing com.* is illegal.
  - // It is perfectly legal but will not help here.
- By importing com.enthu.* and referring to the class as Base.
- By referring to the class as com.enthu.Base.

Explanation

A class or interface can be referred to by using its fully qualified name or its simple name.
Using the fully qualified name will always work, but to use the simple name either the class must be in the same package or it has to be imported.
By importing com.enthu.* all the classes from the package will be imported and can be referred to using simple names.
Importing com.* will not import the subpackage enthu. It will only import the classes in package com.

Problema 7

Consider the following program:

```
public class TestClass{
  public static void main(String[] args){
    String tom = args[0];
    String dick = args[1];
    String harry = args[2];
    System.out.println(harry);
  }
}
```

What will be printed if the program is run using the command line:

java TestClass 111 222 333

- 111
- 222
- 333
- It will throw an ArrayIndexOutOfBoundsException
- None of the above.

Explanation

java and classname are not part of the args array. So tom gets "111", dick gets "222" and harry gets "333".

https://youtu.be/f2BgLQxMilc

Problema 8

Consider the directory structure shown in Image 1 that displays available folders and classes and the code given below:

```
class StockQuote{
  Stock stock;
  public StockQuote(Stock s)  {
  }
  public void store() throws IOException{
     Util.store(stock);
  }
  public double computePrice(){
    return Helper.getPricer(stock).price();
  }
}
```

Assuming that the code uses valid method calls, what statements MUST be added to the above class?

Please Select 4 options:

- package com.enthu.rad.*;
    - // Bad syntax. A package statement can never have a *. It should specify the exact package name
- import com.enthu.*;
- package com.enthu.rad;
    - // Since there is no import statement available for com.enthu.rad package, you must put the given class in com.enthu.rad package so that it will be accessible. Classes of the same package are always available to each other.
- import com.*;
- import java.io.*;
- It is not required to import java.io.* or import java.io.IOException because java.io package is imported automatically.
    - // Since the code is using IOException, the java.io package (or just java.io.IOException class) must be imported. Only java.lang package is imported automatically.

https://youtu.be/iKBFZQ0Dnuo

Problema 9

What will be the output of the following program when it is compiled and run with the command line:

```
java TestClass 1 2 3
```

```
public class TestClass {
    public static void main(String[] args) {
        System.out.println("Values : "+args[0]+args[1]);
    }
}
```

Please select 1 option:

- Values : java TestClass
- Values: TestClass 1
- Values: 12
- Values: 3

// No hay espacios entre string concatenadas

Explanation:

In Java, command line arguments are passed into the program using the String[] parameter to the main method. The String array contains actual parameters and does not include java and the name of the class.

Therefore, in this case, args will point to an array of Strings with 3 elements - "1", "2", and "3". The program prints out only args[0] and args[1], which are 1 and 2.

Problema 10

What will the following code snippet print when compiled and run?

```
byte starting = 3;
short firstValue = 5;
int secondValue = 7
int functionValue =
     (int) (starting/2 + firstValue/2 + (int) firstValue/3 ) + secondValue/2;
     // (int) ( 3/2 + 5/2  + int(5/3))  + int(7/2)
     // (int) ( byte(1.5) +  short(2.5)  +  (1) )   *  (3)
     // (int) (1 + 2 + 1) + (3)  // 7
     System.out.println(functionValue);
```

Explanation:

Remember that whenever both the operands of a mathematical operator (such as / and *) are integral types except long (i.e. byte, char, short, and int), the result is always the integer value that remains after truncating the fractional value. For example, 5/3 is 1.6666 but the result will be 1 after removing the fractional part. Observe that there is no "rounding off" here.

In the expression given in this question, starting, firstValue, and secondValue are of type byte, short, and int respectively. Thus, the above rule is applicable here. Therefore, starting/2 will result in 1, firstValue/2 will result in 2, firstValue/3 will result in 1, and secondValue/2 will result in 3. The expression will therefore be equivalent to:

```
    (int) ( 1 + 2 + 1 ) + 3)
=> (int)(4) + 3
=> 7
```

Although not important for this question, you should remember that the type of the result will be int even if both the operands are of a type that is smaller than an int. Thus, the following will not compile -

```
byte b1 = 1;
byte b2 = 2;
byte b = b1 + b2; //result is of type int, which cannot be assigned directly to a byte
```

You have to use a cast:

```
byte b = (byte) (b1 + b2); //OK now
```

Similarly, when one of the operands is of type long, float, or double and the other operand is of a smaller size, the result will be a long, float, or double respectively.

Problema 11

Consider the following code:

```
 import java.util.ArrayList;
 public class Student{

    ArrayList<Integer> scores;
    private double average;

    public ArrayList<Integer> getScores(){ return scores; }

    public double getAverage(){ return average; }

    private void computeAverage(){
        //valid code to compute average
        average =//update average value
    }
    public Student(){
        computeAverage();
    }
    //other code irrelavant to this question not shown
}
```

What can be done to improve the encapsulation of this class?

Please select 2 options:

- Make the class private
- Make the scores instance field private.
    - //An important aspect of encapsulation is that other classes should not be able to modify the state fields of a class directly. Therefore, the data members should be private (or protected if you want to allow subclasses to inherit the field) and if the class wants to allow access to these fields, it should provide appropriate setters and getters with public access.
- Make getScores() protected.
- Make computeAverage() public.

- Change getScores to return a copy of the scores list:
    - public ArrayList<Integer> getScores(){
    - return new ArrayList(scores);
    - }
- // If you return the same scores list, the caller would be able to add or remove elements from it, thereby rendering the average incorrect. This can be prevented by returning a copy of the list.

https://youtu.be/rGWfJ1srXtw

Problema 12

Encapsulation ensures that ... (Choose 1 option)

- classes are able to inherit functionality from other classes.
- classes expose only certain fields and methods to other classes for access.
- classes designate certain methods to be abstract and let them be implemented by subclasses.
- a method that takes a class X object as a parameter can be passed an object of a subclass of X.

https://youtu.be/DA_7Y24-EAY

Problema 13

Which of the following are benefits of polymorphism? (Select two options)

- It makes the code more reusable.
- It makes the code more efficient.
    - // This option is a bit ambiguous because it is not clear which efficiency is it talking about - execution, memory, or maintenance. Our guess is that it is referring to execution efficiency. It is not true because polymorphism causes a very slight degradation due to dynamic binding at run time.
- It protects the code by preventing extension.
    - // Just the reverse is true. Extension is how polymorphism is achieved.
- It makes the code more dynamic.
    - // Polymophism allows the actual decision of which method is to be invoked to be taken at runtime based on the actual class of object. This is dynamic binding and makes the code more dynamic.

https://youtu.be/tS3hr_Bc1jE

Problema 14

What can be added to the following Person class so that it is properly encapsulated and the code prints 29?

```
class Person{

    //Insert code here

}

public class Employee extends Person{

        public static void main(String[] args) {

            Employee e = new Employee();

            e.setAge(29);

            System.out.println(e.getAge());

        }

}
```

Please select 2 options

```
    private int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
```

```
    protected int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
```

// Protected is not a valid way to encapsulate a field because any class in a package can access the field.

```
    int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
```

// No access modifer to age means it has default access i.e. all the members of the package can access it. This breaks encapsulation.

```
    private int age;
    private int getAge() {
        return age;
    }
    private void setAge(int age) {
        this.age = age;
    }
```

// If you make getAge and setAge private, you cannot call them from Employee class.

```
    private int age;
    public int getAge() {
        return age;
    }
    protected void setAge(int age) {
        this.age = age;
    }
```

// En este caso protected se puede ya que esta dentro del mismo paquete

Explanation:

This is a ambiguous question because it doesn't give all the information. It really depends on the business logic of the class and the whole application whether the accessor methods (and specially the setter) should be public or protected or even private. The field should be private. Expect such questions in the exam.

https://youtu.be/qveHUHJNZrU

Problema 15

What will be result of attempting to compile the following code appearing in TestClass.java file?

```
import java.util.*;
package test;
public class TestClass{
    public OtherClass oc = new OtherClass();
}
class OtherClass{
    int value;
}
```

Select 1 option:

- The class will fail to compile, since the class OtherClass is used before it is defined.
- There is no problem with the code.
- The class will fail to compile, since the class OtherClass must be defined in a file called OtherClass.java
- The class will fail to compile.
- None of the above.

Explanation:

The order is:

package statement.

import statements

class/ interface definitions.

Important point to note here is YOU MUST READ THE QUESTIONS VERY CAREFULLY.

https://youtu.be/gZVBScKDnks

Problema 16

Consider the classes shown below:

```
class A{
   public A() { }
   public A(int i) {   System.out.println(i );    }
}
class B{
   static A s1 = new A(1);
   A a = new A(2);
   public static void main(String[] args){
      B b = new B();
      A a = new A(3);
   }
   static A s2 = new A(4);
}
```

Which is the correct sequence of the digits that will be printed when B is run?

- 1 ,2 ,3 4.
- 1 ,4, 2 ,3
- 3, 1, 2, 4
- 2, 1, 4, 3
- 2, 3, 1, 4

Explanation:

1. All static constants, variables, and blocks. Among themselves the order is the order in which they appear in the code. This step is actually a part of "class initialization" rather than "instance initialization". Class initialization happens only if the class is being used for the first time while being instantiated.

For example, if you have invoked a static method of a class or accessed a static field of that class earlier in the code, you have already used the class and the JVM would have performed initialization of this class at that time. Thus, there wouldn't be any need to initialize the class if you instantiate an object of this class now.

Further, if the class has a superclass, then the JVM performs this step for the superclass first (if the superclass hasn't been initialized already) and only after the superclass is initialized and static blocks are executed, does the JVM proceed with this class. This process is followed recursively up

to the java.lang.Object class.

2. All non static constants, variables, and blocks. Among themselves the order is the order in which they appear in the code.

3. Constructor.

Just like the class initialization, instance initialization also happens for the superclass first. That is, if the class has a superclass, then the JVM takes steps 2 and 3 given above for the superclass first and only after the superclass's instance members are initialized, does the JVM proceed with this class. This process is also followed recursively up to the java.lang.Object class.

https://youtu.be/umGZgbx3zEk

Problema 17

Consider the following code appearing in a file named TestClass.java:

```
class Test{ }  // 1


public class TestClass {


   public int main(String[] args)  { // 2


       double x=10, double y;  // 3
    // No es posible ya que debe separarse con coma sin doublé o con semi
    // double x=10; double y;      // double x=10, y;
       System.out.println[]; // 4


        for(int k =0; k<x; k++){ } //5
        // X puede ser leida como int oo comparada con su valor int
       return 0;
   }
}
```

Which of the lines are invalid?

- // 1 and // 4
- // 3 and // 4
- // 2 and // 4
- // 2 and // 3

Explanation:

// 1 is valid because it is a valid code that declares a class.

// 2 is a valid declaration of a method named main. Although, it is not a correct declaration for the standard main method that can be used to execute the class, but it is a valid method nevertheless.

// 3 is invalid syntax. It can be written as either double x=10; double y; or double x=10, y;

Note that even though x is a double and 10 is an int, it is valid because 10 will automatically be converted to a double. The reverse would not be valid i.e. int x = 10.0; will be invalid.
You need a cast for that: int x = (int) 10.0;

//4 is invalid because println is not a class name. So you cannot create an array of it. println is a method. So it should be written as System.out.println();

//5 is a valid declaration of a for loop.

https://youtu.be/wGiBLtZjqVE

Problema 18

Given the following contents of two java source files:

```
package util.log4j;
public class Logger  {
  public void log(String msg){
      System.out.println(msg);
  }
}
```

and

```
package util;
public class TestClass {
    public static void main(String[] args) throws Exception {
        Logger logger = new Logger();
        logger.log("hello");
    }
}
```

What changes, when made independently, will enable the code to compile and run?

Please select 2 options:

Replace Logger logger = new Logger(); with:

log4j.Logger logger = new log4j.Logger();

// If you are not importing a class or the package of the class, you need to use the class's fully qualified name while using it. Here, you need to use util.log4j.Logger instead of just log4j.Logger:
util.log4j.Logger logger = new util.log4j.Logger();

Replace package util.log4j; with

package util;

// This will put both the classes in the same package and TestClass can then directly use Logger class without importing anything.

Replace Logger logger = new Logger(); with:

util.log4j.Logger logger = new util.log4j.Logger();

// Using a fully qualified class name always works irrespective of whether you import the package or not. In this case, all classes of package util are available in TestClass without any import statement but Logger is in util.log4j. Therefore, the use of fully qualified class name for Logger, which is util.log4j.Logger, makes it work.

Remove package util.log4j; from Logger.

// Remember that you can never access a class that is defined in the default package (i.e. the package with no name) from a class in any other package. So if you remove the package statement from Logger, you can't access it from util package, which is where TestClass is.

Add import log4j; to TestClass.

// This will not help because Logger is in util.log4j package and not in log4j package.

https://youtu.be/9Hr_o7y3nyI

Problema 19

You have written some Java code in MyFirstClass.java file. Which of the following commands will you use to compile and run it. (Assume that the code has no package declaration.)

- javac MyFirstClass.java
- javar MyFirstClass

- javap MyFirstClass.java
- javar MyFirstClass.java

- java MyFirstClass.java
- java MyFirstClass.class

- javac MyFirstClass.java
- javar MyFirstClass.java

- javac MyFirstClass.java
- java MyFirstClass

Explanation

Remember that java code must be written in a file with .java extension. If you have a public type (class, interface, or enum) in the code, the name of the file must be same as the name of that public type.

Compilation and execution of a Java program is two step process. You first need to compile a java file using a Java compiler. Oracle's JDK comes with a compiler. It is contained in the executable file named javac. You will find it in <jdk installation folder>/bin.

javac compiles the source code and generates bytecode in a new file with the same name as the source file but with extension .class. By default, the class file in generated in the same folder but javac is capable of placing it in a different folder if you use the -d flag. [This is just FYI and not required for the exam. -d is a very important and useful flag and we recommend that you read about it even if it is not required for the exam.]

In second step, the Java virtual machine (JVM), aka Java interpreter is invoked to execute the .class file. Oracle's JVM is contained in the executable file named java. It is also present in the same bin folder of JDK installation. It takes the fully qualified name (i.e. name including package) of the class file without extension as a argument.

Problema 20

What is meant by "encapsulation" ?

- There is no way to access member variable.
- There are no member variables.
- Member fields are declared private and public accessor/mutator methods are provided to access and change their values if needed.
    - // Me parece que la respuesta es 'if needed'
- Data fields are declared public and accessor methods are provided to access and change their values.

Explanation

Encapsulation is one of the 4 fundamentals of OOP (Object Oriented Programming).

Encapsulation means that the internal representation of an object is generally hidden from view outside of the object's definition. Typically, only the object's own methods can directly inspect or manipulate its fields. Some languages like Smalltalk and Ruby only allow access via object methods, but most others (e.g. C++ or Java) offer the programmer a degree of control over what is hidden, typically via keywords like public and private.

Hiding the internals of the object protects its integrity by preventing users from setting the internal data of the component into an invalid or inconsistent state. A benefit of encapsulation is that it can reduce system complexity, and thus increases robustness, by allowing the developer to limit the interdependencies between software components.

Problema 21

Given the following code:

```
interface Movable{
   int offset = 100;
   public void move(int dx);
}
interface Growable{
  public void grow(int dy);
}

class Animal implements Movable, Growable{
   public void move(int dx){ }
   public void grow(int dy){ }
}
```

- Animal class illustrates Java's support for multiple inheritance of type.
    - // Se dice que 'type' se refiere a herencia, clases y enums. Por lo que en este caso hace alusión a que se usa la herencia 2 veces, y no multiple (o herencia multiple como en otros casos)
- Animal class illustrates Java's support for multiple inheritance of state.
- Animal class illustrates Java's support for multiple inheritance of type as well as state.
- Animal class illustrates Java's support for multiple implementation inheritance.

Problema 22

Given the following code -

```
public class MyFirstClass{
  public static void main(String[] args){
     System.out.println(args[1]);
  }
}
```

Which of the following commands will compile and then print "hello"?

- javac MyFirstClass
- java MyFirstClass hello hello

- javac MyFirstClass.java
- java MyFirstClass hello hello

// Since the code is printing args[1] i.e. the second parameter, you need to specify "hello" as the second argument. The first argument is ignored by this code. If you do not specify two parameters, this code will throw ArrayIndexOutOfBoundsException because it will be trying to access the second element of an array of size 1.

- javac MyFirstClass
- java MyFirstClass hello

- javac MyFirstClass.java
- java MyFirstClass hello

https://youtu.be/cGgENo5gjLQ

Problema 23

Given the following set of member declarations, which of the following is true?

```
int a;    //  (1)
static int a;    //  (2)
int f( )   { return a; }    //  (3)
static int f( ) { return a; }    //  (4)
```

- Declarations (1) and (3) cannot occur in the same class definition.

- Declarations (2) and (4) cannot occur in the same class definition.
- // A static method can refer to a static field.

- Declarations (1) and (4) cannot occur in the same class definition.
- // Because method f() is static and a is not.

- Declarations (2) and (3) cannot occur in the same class definition.
- // Can an Instance method act acts as a class variable 'YES'
- // IS principle, Instance to static is Ok

- Declarations (1) and (2) cannot occur in the same class definition
- // variable names must be different.

//Static methods do not have 'this' ref, => static methods cannot directly (without a ref) Access instance variables

Explanation:

Local variables can have same name as member variables. The local variables will simply shadow the member variables with the same names.

Declaration (4) defines a static method that tries to access a variable named 'a' which is not locally declared. Since the method is static, this access will only be valid if variable 'a' is declared static within the class. Therefore declarations (1) and (4) cannot occur in the same definition.

https://youtu.be/Hf8oeEajeSg

Problema 24

What will the following code print when run?

```
public class TestClass{
  public static long main(String[] args){
     System.out.println("Hello");
     return 10L;
  }
}
```

- Hello
- It will not print anything.
- It will not compile
- It will throw an Error at runtime.
- None of the above

Explanation

When the program is run, the JVM looks for a method named main() which takes an array of Strings as input and returns nothing (i.e. the return type is void). But in this case, it doesn't find such a method (the given main() method is returning long!) so it gives out the following message:

```
Error: Main method must return a value of type void in class TestClass,
please define the main method as:
public static void main(String[] args)
```

Problema 25

Which of these statements are true?

- A static method can call other non-static methods in the same class by using the 'this' keyword.
- // 'this' reference is not available within a static method.

- A class may contain both static and non-static variables and both static and non-static methods.

- Each object of a class has its own copy of each non-static member variable.

- Instance methods may access local variables of static methods.
- // local variables can only be accessed in the method they are defined. So you cannot access them anywhere outside that method.

- All methods in a class are implicitly passed a 'this' parameter when called.
- // All non-static/instance methods in a class are implicitly passed a 'this' parameter when called

Explanation

'this' is assigned a reference to the current object automatically by the JVM. Thus, within an instance method foo, calling this.foo(); is same as calling foo();

Since there is no current object available for a static method, 'this' reference is not available in static methods and therefore it can only be used within instance methods. For the same reason, static methods cannot access non static fields or methods of that class directly i.e. without a reference to an instance of that class.

Note : you can't reassign 'this' like this:
this = new Object();

https://youtu.be/zKxSrQ5vaMs

Problema 26

Consider the following two classes defined in two .java files.

```
//in file /root/com/foo/X.java
package com.foo;
public class X{
  public static int LOGICID = 10;
  public void apply(int i){
    System.out.println("applied");
  }
}

//in file /root/com/bar/Y.java
package com.bar;
//1  <== INSERT STATEMENT(s) HERE
public class Y{
    public static void main(String[] args){
       System.out.println(X.LOGICID);
    }
}
```

What should be inserted at //1 so that Y.java can compile without any error?

In this code Y is using LOGICID but via or through X.

So we need 'X' and once we get 'X' we'll get LOGICID by hash, supongo que se refiere al *.

Explanation

import static X;

import static com.foo.*;

// Bad syntax. Package import does not use static keyword.

import static com.foo.X.*;

// This static import, although syntactically correct, will not help here because Y is accessing class X in X.LOGICID. Solo accede a LOGICID

import com.foo.*; // Esto devuelve 'x' que e slo que se busca

// This is required because Y is accessing class X. static import of LOGICID is NOT required because Y is accessing LOGICID through X (X.LOGICID). Had it been just System.out.println(LOGICID), only one import statement: import static com.foo.X.*; would have worked.

import com.foo.X.LOGICID;

// Bad Syntax. Syntax for importing static fields is: import static <package>.<classname>.*; or import static <package>.<classname>.<fieldname>;

Problema 27

Which of the following are features of Java?

Some candidates have reported a similar question being asked with a slightly different (and ambiguous) wording:

Which of the following are objected oriented features of Java?

- Every class must have a main method so that it can be tested individually from command line.
- // Testing of a class and execution of a class from command line are two entirely different things. Execution of a class from command line requires the class to have the standard main method but execution of the main method does not mean that the class is tested.

- Every class belongs to a package.
- // This is true because as per Section 7.4.2 of JLS, "A compilation unit that has no package declaration is part of an unnamed package." Thus, if you omit the package statement, the class will belong to the unnamed package. Remember that classes in the unnamed package are accessible only to other classes in the unnamed package. It is not possible to import this unnamed package in classes that belong to a named package.

- A package must have more than one class.
- // A package may have just one class as well.

- A class may inherit from another class. // La Objeta

Problema 28

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

```
//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.

Problema 29

Given the following class, which of these are valid ways of referring to the class from outside of the package com.enthu?

```
package com.enthu;

public class Base{

  // lot of code...

}
```

- Base
- // Only if you import the whole package containing the class or import the class first.

- By importing the package com.* and referring to the class as enthu.Base
- // package 'com' does not contain Base.

- importing com.* is illegal.
- // It is perfectly legal but will not help here.

- By importing com.enthu.* and referring to the class as Base.

- By referring to the class as com.enthu.Base.

Explanation

A class or interface can be referred to by using its fully qualified name or its simple name. Using the fully qualified name will always work, but to use the simple name either the class must be in the same package or it has to be imported.

By importing com.enthu.* all the classes from the package will be imported and can be referred to using simple names. Importing com.* will not import the subpackage enthu. It will only import the classes in package com.

Problema 30

You are writing a class named Bandwidth for an internet service provider that keeps track of number of bytes consumed by a user. The following code illustrates the expected usage of this class -

```
class User{
    Bandwidth bw = new Bandwidth();

    public void consume(int bytesUsed){
        bw.addUsage(bytesUsed);
    }
   ... other irrelevant code
}

class Bandwidth{
    private int totalUsage;
    private double totalBill;
    private double costPerByte;

    //add your code here

    ...other irrelevant code
}
```

Your goal is to implement a method addUsage (and other methods, if required) in Bandwidth class such that all the bandwidth used by a User is reflected by the totalUsage field and totalBill is always equal to totalUsage*costPerByte. Further, that a User should not be able to tamper with the totalBill value and is also not able to reduce it.

Which of the following implementation(s) accomplishes the above?

```
public void addUsage(int bytesUsed){
    if(bytesUsed>0){
        totalUsage = totalUsage + bytesUsed;
        totalBill = totalBill + bytesUsed*costPerByte;
    }
}
```

```
protected void addUsage(int bytesUsed){
        totalUsage += bytesUsed;
        totalBill = totalBill + bytesUsed*costPerByte;
}
```

// There is no validity check for bytesUsed argument. User will be able to tamper will the bill by suppling a negative number for bytesUsed

```
private void addUsage(int bytesUsed){
    if(bytesUsed>0){
        totalUsage = totalUsage + bytesUsed;
        totalBill = totalUsage*costPerByte;
    }
}
```

// If this method is made private, User class will not be able to access it.

```
public void addUsage(int bytesUsed){
    if(bytesUsed>0){
        totalUsage = totalUsage + bytesUsed;
    }
}
public void updateTotalBill(){
    totalBill = totalUsage*costPerByte;
}
```

// This is not a good approach because once the User class calls addUsage() method, totalBill field will not reflect the correct amount unless User also calls updateTotalBill, which means Bandwidth class is now dependent on some other class to keep its internal state consistent with the business logic.

Problema 31

Identify correct option(s)

- Multiple inheritance of state includes ability to inherit instance methods from multiple classes.
- // Methods do not have state. Ability to inherit instance methods from multiple classes is called multiple inheritance of implementation. Default methods introduce one form of multiple inheritance of implementation. A class can implement more than one interface, which can contain default methods that have the same name. However, such a class cannot be compiled. In this case, the implementing class is required to provide its own implementation of the common method to avoid ambiguity.

- Multiple inheritance of state includes ability to inherit instance fields from multiple classes.

- Multiple inheritance of type includes ability to inherit instance fields as well as instance methods from multiple classes.

- Multiple inheritance of type includes ability to implement multiple interfaces and ability to inherit static or instance fields from interfaces and/or classes.

- Multiple inheritance of type includes ability to implement multiple interfaces and/or ability to extend from multiple classes.

Explanation:

Interfaces, classes, and enums are all "types". Java allows a class to implement multiple interfaces. In this way, Java supports multiple inheritance of types.

"State", on the other hand, is represented by instance fields. Only a class can have instance fields and therefore, only a class can have a state. (Fields defined in an interface are always implicitly static, even if you don't specify the keyword static explicitly. Therefore, an interface does not have any state.) Since a class is allowed to extend only from one class at the most, it can inherit only one state. Thus, Java does not support multiple inheritance of state.

Problema 32

Consider the following code:

```
import java.util.ArrayList;

public class Student{

    ArrayList<Integer> scores;
    private double average;

    public ArrayList<Integer> getScores(){ return scores; }

    public double getAverage(){ return average; }

    private void computeAverage(){
        //valid code to compute average
        average =//update average value
    }

    public Student(){
        computeAverage();
    }
    //other code irrelavant to this question not shown
}
```

What can be done to improve the encapsulation of this class?

- Make the class private.

- Make the scores instance field private.
- An important aspect of encapsulation is that other classes should not be able to modify the state fields of a class directly. Therefore, the data members should be private (or protected if you want to allow subclasses to inherit the field) and if the class wants to allow access to these fields, it should provide appropriate setters and getters with public access.

- Make getScores() protected.

- Make computeAverage() public.

- Change getScores to return a copy of the scores list:
    - public ArrayList<Integer> getScores(){
    - return new ArrayList(scores);
    - }
- If you return the same scores list, the caller would be able to add or remove elements from it, thereby rendering the average incorrect. This can be prevented by returning a copy of the list.

Problema 33

Given the following contents of two java source files:

```
package util.log4j;
public class Logger  {
  public void log(String msg){
      System.out.println(msg);
  }
}
```

and

```
package util;
public class TestClass {
    public static void main(String[] args) throws Exception {
        Logger logger = new Logger();
        logger.log("hello");
    }
}
```

What changes, when made independently, will enable the code to compile and run?

Explanation:

You had to select 2 options

- Replace Logger logger = new Logger(); with: log4j.Logger logger = new log4j.Logger();
- // If you are not importing a class or the package of the class, you need to use the class's fully qualified name while using it. Here, you need to use util.log4j.Logger instead of just log4j.Logger: util.log4j.Logger logger = new util.log4j.Logger();

- Replace package util.log4j; with package util;
- // This will put both the classes in the same package and TestClass can then directly use Logger class without importing anything.

- Replace Logger logger = new Logger(); with: util.log4j.Logger logger = new util.log4j.Logger();
- // Using a fully qualified class name always works irrespective of whether you import the package or not. In this case, all classes of package util are available in TestClass without any import statement but Logger is in util.log4j. Therefore, the use of fully qualified class name for Logger, which is util.log4j.Logger, makes it work.

- Remove package util.log4j; from Logger.
- // Remember that you can never access a class that is defined in the default package (i.e. the package with no name) from a class in any other package. So if you remove the package statement from Logger, you can't access it from util package, which is where TestClass is.

- Add import log4j; to TestClass.
- // This will not help because Logger is in util.log4j package and not in log4j package.

https://youtu.be/9Hr_o7y3nyI

Problema 34

Given:

```
public class Triangle{
    public int base;
    public int height;
    public double área;
    public Triangle(int base, int height){
        this.base = base; this.height = height;
        updateArea();
    }
    void updateArea(){
        area = base*height/2;
    }
    public void setBase(int b){ base  = b; updateArea(); }
    public void setHeight(int h){ height  = h; updateArea(); }
}
```

The above class needs to protect an invariant on the "area" field. Which three members must have the public access modifiers removed to ensure that the invariant is maintained?

You had to select 3 options

- the base field
- the height field
- the area field
- the Triangle constructor
- the setBase method
- the setHeight method

An invariant means a certain condition that constrains the state stored in the object. For example, in this case the value of the area field of the Triangle must always be consistent with its base and height fields. Thus, it should never have a value that is different from base*height/2. If you allow other classes to directly change the value of base, height, or area, using direct field access, the area field may not contain the correct area thereby breaking the invariant. To prevent this inconsistency from happening, you need to prohibit changing the instance fields directly and instead permit the changes only through the setter method because these methods call the updateArea method and keep the area and base and height consistent.

https://youtu.be/6j98ij1pE8s

Problema 35

You are asked to develop an application for a car rental company. As a part of that, you are given the following requirements -

- Implement three classes - Car, SUV, and MiniVan, where the Car class is the super class of SUV as well as MiniVan.
- Implement method int getDailyRate() that returns the daily price of the car.
- Implement method void printDetails() that prints the details of the car.

Which of the following definition of Car class adds a valid layer of abstraction to the class hierachy?

You had to select 1 option

- public abstract class Car{ public abstract int getDailyRate(); public void printDetails(){ // code for printing details goes here } }
- // Since Car class does not know the details of SUV and MiniVan, you can't provide the code for them in this class. Therefore, you should make this method abstract.

- public abstract class Car{ public int getDailyRate(); public void printDetails(); }
- // This is invalid because of the lack of abstract keyword on the methods. This will not compile and is, therefore, an obviously wrong answer.

- public abstract class Car{ public abstract int getDailyRate(); public abstract void printDetails(); }
- // As per the given information, Car could be an abstract class with two methods. You need to make these two methods abstract so that concrete classes such as SUV and MiniVan will be forced to provide appropriate implementations of these methods.

- public abstract class Car{ public abstract int getDailyRate(); public abstract void printDetails(){ // code for printing details goes here } }
- // A method that has code cannot be abstract and vice-versa. This will not compile and is, therefore, an obviously wrong answer

Explanation:

The problem statement is very ambiguous and there are multiple valid implementations. You will need to draw clues from the options and select the best option by eliminating options that are obviously wrong. Expect such questions in the exam.

https://youtu.be/6OfYj1wskyM

Problema 36

The options below contain the complete contents of a file.

Which of these options can be run with the following command line once compiled? `java main`

You had to select 1 option

- //in file main.java
- class main {
- public void main(String[] args) {
- System.out.println("hello");
- }
- }
- // The main method should be static.

- //in file main.java
- public static void main(String[] args) {
- System.out.println("hello");
- }
- // You cannot have a method on its own. It must be a part of a class.

- //in file main.java
- public class anotherone{ }
- class main {
- public static void main(String[] args) {
- System.out.println("hello");
- }
- }
- // A public class must exist in a file by the same name. So this code is invalid because anotherone is a public class but the name of the file is main. It would have been valid if the name of the file were anotherone.java. A non public class may exist in any file. This implies that there can be only one public class in a file.

- //in file main.java
- class anothermain{
- public static void main(String[] args) {
- System.out.println("hello2");
- }
- }

- class main {
- public final static void main(String[] args) {
- System.out.println("hello");
- }
- }
- // Observe that there is no public class in file main.java. This is ok. It is not necessary for a java file to have a public class. The requirement is that if a class (or enum) is public then that class (or enum) must be defined in a file by the same name and that there can be only one public class (or enum) in a file. class main's main method will be executed. final is a valid modifier for the standard main method. Note that final means a method cannot be overridden. Although static methods can never be overridden (they can be hidden), making a static method final prevents the subclass from implementing the same static method.

Explanation:

Observe that the given code does not follow the standard Java naming convention. The class names should start with a capital letter.

There are questions in the exam that contain similar non-conventional and confusing names and that is why we have kept a few questions like that in this question bank.

https://youtu.be/ufjTCm9FxHo

Problema 37

Given the following program, which statement is true?

```
class SomeClass{

   public static void main( String args[ ] ){

      if (args.length == 0 ){

         System.out.println("no arguments") ;

      }

      else{

         System.out.println( args.length + " arguments") ;

      }

   }

}
```

You had to select 1 option

- The program will fail to compile.
- The program will throw a NullPointerException when run with zero arguments.
- The program will print no arguments when called with zero arguments and 1 arguments when called with one argument.
- // The word java and class name are not a part of the argument list.
- The program will print no arguments and 2 arguments when called with zero and one arguments.
- The program will print no arguments and 3 arguments when called with zero and one arguments.
- // When the program is called with no arguments, the args array will be of length zero.

Explanation:

When the program is called with no arguments, the args array will be of length zero. Unlike in C/C++, args[0] is not the name of the program or class. This is because the name of the class is always the same as defined in the java file. So there is no need for passing its name as an argument to main method.

Problema 38

Which of the given options can be successfully inserted at line 1....

```
//line 1
   public class A{
   }
```

You had to select 3 options

- import java.lang.*;
- // Although this package is automatically imported, it is not an error to import it explicitly.

- package p.util;
- // It is a perfectly valid package statement.

- public class MyClass{ }
- // There can be only 1 "public" class within package scope in a file. You can have additional inner classes that are public though.

- abstract class MyClass{ }
- // You can have more than one classes in a file but at most one of them can be public.

Problema 39

Consider the following code appearing in a file named TestClass.java:

```
class Test{ }  // 1


public class TestClass {


   public int main(String[] args)  { // 2
       double x=10, double y;  // 3
       System.out.println[]; // 4
        for(int k =0; k<x; k++){ } //5
       return 0;
   }
}
```

Which of the lines are invalid?

You had to select 1 option

- // 1 and // 4
- // 3 and // 4
- // 2 and // 4
- // 2 and // 3

// 1 is valid because it is a valid code that declares a class.

// 2 is a valid declaration of a method named main. Although, it is not a correct declaration for the standard main method that can be used to execute the class, but it is a valid method nevertheless.

// 3 is invalid syntax. It can be written as either double x=10; double y; or double x=10, y; Note that even though x is a double and 10 is an int, it is valid because 10 will automatically be converted to a double. The reverse would not be valid i.e. int x = 10.0; will be invalid. You need a cast for that: int x = (int) 10.0;

//4 is invalid because println is not a class name. So you cannot create an array of it. println is a method. So it should be written as System.out.println();

//5 is a valid declaration of a for loop.

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

- [x] String s;
- [ ] String s = 'asdf';

A string must be enclosed in double quotes ".

- [ ] String s = 'a';

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

- [x] 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.

- [x] 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.

- [x] 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.

☑ 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.

☑ 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

- [x] `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"));`
- [x] `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

- [x] 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.

- [x] 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.

- [x] 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.

- [x] 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.

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

- [x] 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

- [x] 4
- [x] 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

- [x] 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.
- [x] 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

- [x] If run with an argument of 'false', it will print 'False False'
- [ ] If run with an argument of 'false', it will print 'True True'
- [x] If run with an argument of 'true', it will print 'True False'
- [x] 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 ;`
- [x] `s *= b ;`

All compound assignment operators internally do an explicit cast.

- [ ] `c = c + b ;`

c + b returns an int

- [x] `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
- [x] 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