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.