1. **Ejercicio de análisis**

   a) Explique por qué la clase Instrumento debe definirse como abstracta y qué la diferencia de una clase normal, en el ejemplo siguiente:

```
public abstract class Instrumento
{
    private String tipo;
    public Instrumento(String tipo) { this.tipo = tipo; }
    public String getTipo()     { return tipo;}

    public abstract void Tocar();
    public abstract void Afinar();
}

public class Saxofon extends Instrumento
{
    public Saxofon(String tipo) { super(tipo); }
    public void Tocar() { System.out.println("Tocando Saxofon");}
    public void Afinar() {System.out.println("Afinando Saxofon");}
}

public class Guitarra extends Instrumento {
    public Guitarra(String tipo) { super(tipo); }
    public void Tocar() { System.out.println("Tocando Guitarra");}
    public void Afinar() {System.out.println("Afinando Guitarra");}
}
```

   R/: La clase instrumento se debe de definir como abstracta ya que nos permite reutilizar los métodos en sus clases hijas (ya que cada uno de los instrumentos se deben poder tocar y afinar), en cada uno de ellos realizar diferentes procesos, la mayor diferencia entre la abstracta y normal es que no se encuentra definidos los métodos y no se pueden crear instancias de la clase instrumento en las clases hijas solo llamar sus métodos para reutilizarlos

b) Elabore una clase denominada Piano heredada de Instrumento, añada un constructor y redefina para ella los métodos Tocar y Afinar.

R/: public class Piano extends Instrumento {

```
public Piano(String tipo) { super(tipo); }

public void Tocar () {System.out.println("Tocando Piano");}

public void Afinar () {System.out.println("Afinando Piano");}
```

}

c) ¿Señale cuál de las siguientes líneas no se pueden ejecutar y por qué?

```
public class Test {
    public static void main(String[] args) {
        Instrumento x;
        x = new Saxofon("xxxxx");
        x = new Guitarra("xxxxx");
    }
}
```

R/: Todas las líneas de código pueden ser ejecutadas sin ningún problema

d) ¿Qué imprime el siguiente programa?

```
public class Test {
    public static void main(String[] args) {
        Instrumento x;
        x = new Saxofon("xxxxx");   x.Tocar();
        x = new Guitarra("xxxxx");  x.Tocar();
    }
}
```

R/: Tocando Saxofon; Tocando Guitarra

2. Ejercicio de código en el repositorio

a. ¿Qué sucede si se define el método *explotar()* de la clase *Estrella* como se indica a continuación? Explique su respuesta.

```
abstract class Estrella extends ObjetoAstronomicoExtraSolar {
    abstract void explotar() {
        System.out.println("Estrella explotar");
    }
    int a = super.getID();
    public void tipoCuerpol() {
        System.out.println("Simple " + a);
    }
}
```

R/: Estaría mal, pues los métodos abstractos de las clases abstractas deben de ser declarados mas no definidos

b. ¿Qué significa que los métodos *tipoCuerpo2()* y *getID()* de la clase *ObjetoAstronomicoExtraSolar*, no se definan como *abstract*? ¿Podría considerarse esta situación un error? Explique.

R/: No sería un error, simplemente indica que las subclases no necesitan redefinir estos métodos ya que simplemente los heredan

c. Si se define como abstracta la clase *ObjetoAstronomicoExtraSolar*, como se indica a
continuación, ¿puede considerarse un error definir una clase abstracta sin métodos abstractos? Explique.

```
abstract class ObjetoAstronomicoExtraSolar {
    private int ID;

    public void tipoCuerpo2() {
        System.out.println("Extrasolar");
    }

    public int getID() {
        return this.ID;
    }
}
```

R/: No seria un error simplemente las subclases heredan los métodos públicos y no podrán hacer uso de la redefinición de ningún método ya que no hay métodos abstractos

d. Explique por qué el arreglo *oa* (línea 19) hace referencia a una clase abstracta y sin embargo, en la línea 25 se invoca el método correspondiente a cada clase derivada.

R/: Ya que cuando se crea el arreglo se indica que todos los elementos que van a entrar son de tipo ObjetoAstronomicoExtraSolar que es una clase abstracta, y luego hace el llamado a un método correspondiente de una clase derivada ya que

se crean objetos de tipo Galaxia, Nova, SuperNova y este método es uno abstracto que se redefine en cada una de las subclases

e. ¿Por qué la línea 29 imprime “Soy una Super Nova” sabiendo que el arreglo *oa* en esa posición fue inicializado con un objeto de tipo *Galaxia*?

   R/: Ya que el apuntador de ese objeto en la línea anterior línea 28 se le asignó un objeto de tipo SuperNova, en lugar del objeto con el que se había creado objeto el cual era Galaxia

f. ¿Por qué en la clase *Estrella* no se define el método *descripcion()* si la superclase lo está solicitando, ya que en este método descripción en abstracto?

   R/: Porque la clase estrella no lo necesita, aparte de que no se le obliga a utilizar todos los métodos de la clase abstracta debe de utilizar al menos uno de los métodos anteriores, lo que si se debe tener en cuenta que en las subclases siguientes si se utilice por lo menos una vez el método abstracto restante.

g. ¿Qué sucede si el método *tipoCuerpo1()* de la clase *Galaxia* se define como privado? ¿Por qué se genera error?

   R/: Esto generaría error ya que los niveles solo se pueden amplificar no generalizar de uno menos accesible a uno mas accesible no al contrario y como se puede evidenciar en el código el método tipoCuerpo1 es de nivel de accesibilidad public.

h. ¿Por qué la clase *Nova* no define el método *tipoCuerpo1()*? ¿Se podría definir? Si lo define, ¿qué interpreta de esta situación?

   R/: No se define al no ser necesario porque para una subclase que a la vez es abstracta, de la super clase Estrella que a su vez es abstracta no se le obliga a sobrescribe ningún método de la super clase y si solo sobrescribe uno tampoco había ningún problema; Se podría definir tranquilamente y se tomaría como una sobreescritura por herencia.

i. ¿Qué imprime la línea 9? ¿Por qué se puede llamar al método *toString()* si la clase *Galaxia* no lo define y su papá *ObjetoAstronomicoExtraSolar* tampoco?

   R/: Imprimiría lo que está por defecto en la impresión en la clase, y esto se puede realizar gracias a que la superclase Object lo tiene predeterminado.

j. ¿Por qué en la línea 11 se puede crear un puntero obN de tipo *ObjetoAstronomicoExtraSolar* si esta es una clase abstracta?

R/: El puntero puede ser de una clase abstracta, lo que no se puede es crear un objeto de una clase abstracta y lo que se está haciendo ahí simplemente es creando el puntero

k. ¿Las siguientes instrucciones (instrucciones en las líneas B y C) son válidas? Explique.

```
A. Nova nova = new Nova();
B. ObjetoAstronomicoExtraSolar ob = new ObjetoAstronomicoExtraSolar();
C. ObjetoAstronomicoExtraSolar oa = nova;
```

R/: La instrucción de la línea B es totalmente incorrecta no se puede crear objetos(instancias) de una clase abstracta. Por el contrario la instrucción de la línea C si es correcta ya que simplemente indica que el apuntador de tipo *ObjetoAstronomicoExtraSolar* será puntero de un objeto ya creado que es de tipo Nova

l. Explique por qué (ver código a continuación) la siguiente instrucción en la línea B es correcta y la instrucción en la línea C es incorrecta. Omitiendo la instrucción en la línea C, ¿qué se imprime por pantalla? Explique su respuesta.

```
A. Nova nova = new Nova();
B. ObjetoAstronomicoExtraSolar oa = nova;
C. oa.explotar();
D. ((Nova) oa).explotar();
```

R/: En el caso de la instrucción de la línea B es correcta porque solo se está indicando que el puntero de tipo clase abstracta señala a un objeto ya creado de tipo clase Nova, diferente es el caso en la línea C que es incorrecto ya que el tipo del puntero de ese objeto no tiene definido este método; Omitiendo la línea C la línea D imprimiría “Boom!” ya que se le cambia el tipo al apuntador pasa a ser de tipo Nova

m. ¿Por qué la línea 15 imprime *true*? ¿Para cualquier objeto que se cree siempre imprimirá lo mismo? ¿Qué imprimen las siguientes líneas? ¿Por qué?

```
obN = null;
System.out.println(obN instanceof Object);
System.out.println("" + obN instanceof Object);
```

R/: Siempre imprimirá true ya que a pesar de ser un apuntador de tipo *ObjetoAstronomicoExtraSolar* este siempre va a heredar de Objet, las siguientes líneas imprimirán False y False ya que obN no es de esos tipos

n. Agregue el siguiente constructor. ¿Se genera error? ¿Se pueden colocar constructores a clases abstractas? ¿Qué sentido tiene?

```
public ObjetoAstronomicoExtraSolar() {
   this.ID = 4;
   this.tipoCuerpo2();
}
```

R/: No generar error agregarles constructores a clases abstractas se puede realizar sin ningún problema, esto cobra sentido al momento que las clases hijas necesiten de atributos inicializados que se crean en el constructor de la clase abstracta

o. Suponga que agrega la clase *EnanaBlanca* como se indica a continuación. ¿Qué se puede concluir de esta nueva situación? ¿Qué errores se presentan y cómo podría corregirlos?

```
class EnanaBlanca extends Estrella {
   void agotarCombustible() {
       System.out.println("Enana blanca muere");
   }
}
```

R/: Esta sería una nueva clase que hereda de la clase Estrella, pero tiene grandes errores los cuales la pedirían la ejecución de código porque no utiliza ninguno de los dos métodos de la subclase debe de sobre escribir por lo menos uno de ellos, para corregir este error podría sobrescribir los métodos explotar() y el método descripción(), también podría poner la clase abstracta y sobrescribir uno de los dos o ninguno