a) Explique por qué la clase Instrumento debe definirse como abstracta y qué la diferencia de una clase normal

Es necesario que se defina como abstracta pues tiene métodos abstractos y se diferencia de una clase normal en que no se puede instanciar.

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

```
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");
    }
}
```

Todas se pueden ejecutar, saxofón y guitarra son clases normales, y es valido poner un apuntador de tipo abstracto, total se puede hacer apuntar a los subtipos.

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();
    }
}
```

Imprime “Tocando Saxofón” y luego “Tocando Guitarra”

2. 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);
    }
}
```

El codigo no compila pues los metodos abstractos no pueden contener su definición.

2.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.

Esto significa que, si hay una clase normal que extienda a ObjetoAstronomicoExtraSolar, podemos usar estos métodos en ella y no hay necesidad de que la subclase los defina, esto no es un error, las clases abstractas pueden tener métodos implementados, siempre y cuando no se marquen como abstractos.

2.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;
    }
}
```

Esto no es un error, el código compila pues no es necesario que una clase abstracta tenga métodos abstractos. Además, puede sernos útil si

conceptualmente tiene sentido que no se instancie un ObjetoAstronomicoExtraSolar.

2.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.

oa contiene subtipos de ObjetoAstronomicoExtraSolar (Pues no puede haber instancias de ObjetoAstronomicoExtraSolar), estos subtipos definen el método descripción, entonces se ejecutan los métodos de las subclases mediante ligadura dinámica.

2.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?

Es cierto que en primer lugar asignamos al espacio 0 una Galaxia, pero en la línea 28 hacemos que haga referencia a lo mismo que oa[2], que es una Supernova.

2.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?

Estrella se puede quedar con descripción como método abstracto pues ella es abstracta también.

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

Porque ese método en la superclase ObjetoAstronomicoExtraSolar tiene visibilidad por defecto, y no se permite que la subclase Galaxia ponga una visibilidad más privada.

2.h) ¿Por qué la clase Nova no define el método tipoCuerpo1()? ¿Se podría definir? Si lo define, ¿qué interpreta de esta situación?
Porque su superclase directa, Estrella, ya lo define y por tanto se usa su definición, además, la clase puede no ser abstracta porque define directa o indirectamente los métodos abstractos de su padre.

2.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?

toString hace parte de Object del cual hereda implícitamente ObjetoAstronomicoExtraSolar, y se usa esa definición.

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

Es valido crear un puntero de tipo abstracto, pues podemos usarlo para apuntar a una instancia de una subclase.

2.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;
```

B no es valido pues no podemos crear una instancia de ObjetoAstronomicoExtraSolar, una clase abstracta.
C es valido pues solo le damos el tipo abstracto al apuntador, el valor es un subtipo que no es abstracto.

2.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();
```

B es correcta pues el apuntador puede ser de tipo abstracto y es válido que apunte a un subtipo de su tipo.
C no es correcta pues si bien usaríamos ligadura dinámica, debe existir el método en la clase del apuntador, y ObjetoAstronomicoExtraSolar no tiene el método explotar.
D, ignorando C, imprime "Boom!"

2.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);
```

La línea 15 imprime True pues nova hereda de Estrella y a su vez Estrella hereda de ObjetoAstronomicoExtraSolar que a su vez hereda de Object. Y si, para cualquier objeto que se cree imprimirá lo mismo.
Las líneas imprimen false y true en ese orden, pues obN no es un objeto, pero un String lo es, luego obN implícitamente se convierte en un String para concatenarse, al final quedando una expresión String instanceof Object, y un String es un objeto.

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

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

No se genera error, si se pueden poner constructores a clases abstractas pues este nos puede servir para inicializar atributos de la clase abstracta, llamar métodos no abstractos o métodos abstractos para llamar a la versión de la subclase.

2.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?

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

Tenemos problemas pues EnanaBlanca no es abstracta, pero lo quedan por definir explotar y descripción, podemos solucionarlo de esta forma:

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

    void descripcion() {
        System.out.println("Soy una Enana Blanca");
    }

    void explotar(){
        System.out.println("Boom!");
    }
}
```