1. **Ejercicio de análisis**

a) La clase Instrumento debe definirse como abstracta ya que las clases que heredan de ella comparten métodos o "comportamientos" que son comunes pero que son implementados por cada subclase de manera específica, se diferencia de una clase normal al no poder ser instanciada, puede tener métodos abstractos y actúa como una "plantilla".

```
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");}
}
```

b) Clase Piano heredada de Instrumento, con un constructor y con 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) Las líneas que no se pueden ejecutar son: ninguna, todas se pueden ejecutar. El código está implementado correctamente, x es una variable de tipo Instrumento en la cual se crean primero un objeto Saxofon y luego otro de tipo Guitarra.

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

d) El siguiente programa imprime:

Tocando Saxofon
Tocando Guitarra

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

2. **Ejercicio de código en el repositorio**

a. Si se define el método *explotar()* de la clase *Estrella* como se indica a continuación, ese método abstracto estaría definido de forma incorrecta porque no puede tener un cuerpo, que un método sea abstracto implica que la implementación debe ser proporcionada por una subclase.

```
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);
   }
}
```

b. Los métodos *tipoCuerpo2()* y *getID()* de la clase *ObjetoAstronomicoExtraSolar* no se definen como *abstract*, eso es correcto porque las clases abstractas pueden implementar también métodos normales, eso no genera una situación de error porque pueden ser heredados por las subclases tal como están o sobreescribirlos si es necesario.
c. Si se define como abstracta la clase *ObjetoAstronomicoExtraSolar*, como se indica a continuación, se puede decir que no es un error definir una clase abstracta sin métodos abstractos, sin embargo es cuestionable en cuestión de diseño y utilidad.

```
abstract class ObjetoAstronomicoExtraSolar {
    private int ID;

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

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

d. El arreglo *oa* en la línea 19 hace referencia a una clase abstracta y sin embargo, en la línea 25 se invoca el método correspondiente a cada clase derivada, esto es posible porque en cada posición del arreglo se encuentran objetos de diferentes clases, galaxia, nova y supernova, al iterar sobre el arreglo se llama al método descripcion( ) definido en cada una de las subclases.
e. En la línea 29 imprime “Soy una Super Nova” porque el arreglo *oa* en esa posición fue inicializado con un objeto de tipo *Galaxia*, que luego (en la linea 28) pasó a apuntar a un objeto de tipo SuperNova cuyo método descripción fue llamado en la línea mencionada.
f. En la clase *Estrella* no se define el método *descripcion()* a pesar de que la superclase lo está solicitando, ya que es también una clase abstracta, no esta obligada a implementar dicho método y puede delegar la responsabilidad de implementar esos métodos a sus subclases.
g. Si el método *tipoCuerpo1()* de la clase *Galaxia* se define como privado este método no podrá ser accedido por fuera de la clase, lo cual no tendria mucho sentido y generará un error cuando se intente llamar desde la clase ObjTaller7.
h. La clase *Nova* no define el método *tipoCuerpo1()*porque lo hereda de la clase Estrella, se podría redefinir si así se desea, esto implicaría que las subclases pueden adaptarse a sus propias necesidades mientras siguen respetando la estructura general de la clase base.
i. La línea 9 imprime: Galaxia, que es el nombre de la clase, seguido de un @ y el valor hash del objeto en formato hexadecimal, se puede llamar al método *toString()* a pesar de que la clase *Galaxia* no lo define y su papá *ObjetoAstronomicoExtraSolar* tampoco porque el método toString se hereda de Object.
j. En la línea 11 se puede crear un puntero obN de tipo *ObjetoAstronomicoExtraSolar* , porque a pesar de que sea una clase abstracta, se pueden declarar variables de su mismo tipo y usarlas para referenciar objetos de sus subclases.

k. La instrucción en la línea B no es correcta ya que quiere instanciar un objeto de una clase abstracta lo cual no es posible, por otra parte, la instrucción en la línea C es válida porque crea un apuntador oa de tipo ObjetoAstronómico que apunta a una subclase del mismo, un objeto de tipo Nova creado previamente.

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

l. La siguiente instrucción en la línea B es correcta porque sucede igual que en el caso anterior, el apuntador de tipo de la superclase apunta al objeto de una subclase, y la instrucción en la línea C es incorrecta debido a que por ligadura dinámica se verifica que el apuntador cuente con dicho método lo cual no sucede y por lo cual no se puede ejecutar dicha línea; omitiendo la instrucción en la línea C, se imprime por pantalla "Boom!" debido al cast explícito del apuntador oa, siguiendo el principio de ligadura dinámica, ejecuta el método explotar de la clase Nova.

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

m. La línea 15 imprime *true* porque obN es un apuntador de tipo ObjetoAstronomicoExtraSolar, que si bien apunta a un objeto de tipo Nova, por el principio de polimorfismo Nova es Estrella también. Para cualquier objeto que se cree siempre imprimirá lo mismo, porque todo objeto que se cree hereda de object, es decir es "instanceof Object". Lo que imprimen las siguientes líneas es false, porque si el objeto referenciado por la variable es null instanceof devuelve false porque no hereda de object y true porque null que representa "nada" más un String será igual a un String, que a su vez, es instanceof Object.

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

n. Añadiendo el siguiente constructor no se genera error ya que es posible colocar constructores a clases abstractas, aunque no se puede crear instancias directas de una clase abstracta, el constructor de una clase abstracta puede ser invocado por las subclases que heredan de ella, así, puede ser útil para inicializar variables o propiedades comunes que serán compartidas por todas las subclases.

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

o. Suponiendo que se agrega la clase *EnanaBlanca* como se indica a continuación, se puede decir de esta nueva situación que se está creando una nueva clase que hereda de estrella, sin embargo, presenta diferentes errores, como que debería implementar los métodos abstractos explotar y descripcion, de lo contrario debería ser definida como una clase abstracta.

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