```
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.List;
public class Ejemplos {
    public static void main(String[] args) {
        Animal uno = new Animal();
        Animal dos = new Dog();
        uno.makeSound();
        dos.makeSound();
        Dog tres = (Dog) new Animal();
        tres.makeSound();
    }
}
class Animal {
    void makeSound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Wau Wau");
    }
}
```

1) Animal sound Wau Wau compilation error
2) Comilation Error
3) Animal sound Wau Wau Animal sound
4) Animal sound

El código no compilará debido al cast inseguro. La línea Dog tres = (Dog) new Animal(); intentará realizar un cast inválido, lo que provoca un error en tiempo de compilación.

```
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.List;
import java.lang.*;
public class Ejemplos {
    public static void main(String[] args) {
        Cambios uno=new Cambios();
        int x=1;
        String hola="hola";
```

```
        StringBuilder hola2=new StringBuilder("hola2");
        Integer x2=4;
        uno.makeSound(x, hola);
        uno.makeSound(x2, hola2);
        System.out.println("Cambios?:
"+x+","+hola+","+x2+","+hola2);
      }
}
class Cambios{
      void makeSound(int x, String s) {
          s="cambiando string";
          x=5;
      }
      void makeSound(Integer x,StringBuilder s) {

        x=9;

        s=s.delete(0,s.length());
       }

   }
```

1) Compilation error
2) Cambios?: 1,hola,4,
3) Cambios?: 1,hola,4,hola2
4) Cambios?: 5,cambiando string,9,

x en main sigue siendo 1 porque x se pasa por valor en el primer método.

hola en main sigue siendo "hola" porque las cadenas en Java son inmutables, y el cambio solo se aplica localmente.

x2 en main sigue siendo 4 porque x2 es un objeto Integer y el valor de los objetos Integer no cambia con asignaciones locales.

hola2 en main es modificado por el método makeSound para tener un contenido vacío, por lo que la impresión será "".

```
interface i1{
 public void m1();

   }

interface i2 extends i1 {
```

```
        public void m2();
    }
    class animal implements i1,i2 {
        //¿Qué métodos debería implementar la clase animal en este
espacio?


    }
```

1) solo m1
2) m1 y m2
3) ninguno
4) error compilación

La clase animal debe implementar ambos métodos m1() y m2() para cumplir con las interfaces que está implementando.

```
public class Main {
    public static void main(String[] args) {

            Padre objetoPadre = new Padre();

            Hija objetoHija = new Hija();

            Padre objetoHija2 = (Padre) new Hija();

            objetoPadre.llamarClase();

            objetoHija.llamarClase();

objetoHija2.llamarClase();

            Hija objetoHija3 = (Hija) new Padre();

            objetoHija3.llamarClase();

    }

}

public class Hija extends Padre {

    public Hija() {

        // Constructor de la clase Hij

    }

    @Override

    public void llamarClase() {
```

```
        System.out.println("Llame a la clase Hija");
    }
}
public class Padre {
    public Padre() {
        // Constructor de la clase Padre
    }
    public void llamarClase() {
        System.out.println("Llame a la clase Padre");
    }
}
```

Resultado:

a) Llame a la clase Padre
Llame a la clase Hija
Llame a la clase Hija
Error: java.lang.ClassCastException
b) Llame a la clase Padre
Llame a la clase Hija
Llame a la clase Hija
Llame a la clase Hija
b) Llame a la clase Padre
Llame a la clase Hija
Llame a la clase Hija
Llame a la clase Padre
Hija objetoHija3 = (Hija) new Padre();

Este es un cast incorrecto que lanzará una ClassCastException en tiempo de ejecución porque new Padre() no es una instancia de Hija. Por lo tanto, no se llegará a la llamada a llamarClase() después de este cast fallido.

```
interface Movable {
   void move();
}
abstract class Vehicle {
  abstract void fuel();
}
class Car extends Vehicle implements Movable {
   void fuel() {
```

```
        System.out.println("Car is refueled");
    }
   public void move() {
        System.out.println("Car is moving");
    }
}
public class Main {
    public static void main(String[] args) {
        Vehicle myCar = new Car();
        myCar.fuel();
         ((Movable) myCar).move();
    }
}
```

```
class Animal {
    void makeSound() throws Exception {
        System.out.println("Animal makes a sound");
    }
}
class Dog extends Animal {
    void makeSound() throws RuntimeException {
        System.out.println("Dog barks");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        try {
            myDog.makeSound();
         } catch (Exception e) {
            System.out.println("Exception caught");
         }
    }
}
```

Cuál sería la salida en consola al ejecutar este código?

1- Dog barks

2- Animal makes a sound

3- Exception caught

4- Compilation error

myDog es una instancia de Dog, referenciada como Animal.
El método makeSound() en Dog lanza una RuntimeException, que es una subclase de Exception. Sin embargo, en este caso, el tipo de excepción lanzada (RuntimeException) no es una excepción revisada y no necesita ser declarada en el método. Dado que Dog

sobrescribe makeSound() sin declarar throws Exception, no hay un problema en el tiempo de ejecución.

```
import java.util.*;
import java.lang.*;
import java.io.*;
class Main {
    public static void main(String[] args) {
        String str = "1a2b3c4d5e6f";
        String []splitStr = str.split("//D");

        for(String elemento : splitStr){
            System.out.println(elemento);
        }
    }
}
```