Programación Concurrente -2023
Práctico Repaso del Paradigma Orientado a Objetos en JAVA

***Repasemos**:*
Cuando se crea una clase, se crea una plantilla donde se definen ***atributos*** *y* ***métodos.***
Cuando se crea un objeto, ***se instancia una clase,*** *mediante el operador* ***new.***
Todos los objetos de la misma clase tienen los mismos atributos y métodos.
Los valores concretos de cada atributo de cada objeto pueden ser diferentes y conforman su estado.

***Repasemos**:*
La herencia nos permite definir una clase como una ampliación de otra.
Una superclase o clase padre, es una clase que es extendida por otra clase, o que otras clases heredan.
Una subclase, clase hija, es una clase que extiende o amplía a otra clase. Hereda todos los campos y los métodos de la superclase.
La herencia es un mecanismo que nos ofrece una solución al problema de duplicación de código.
La característica esencial de esta técnica es que necesitamos describir las características comunes sólo una vez.
La herencia también se denomina relación «es-un».
La razón de esta nomenclatura radica en que la subclase es una especialización de la superclase.
La herencia nos permite crear dos clases que son bastante similares evitando la necesidad de escribir dos veces la parte que es idéntica.
Más de una subclase puede heredar de la misma superclase y una subclase puede convertirse en la superclase de otras subclases.
Las clases que están vinculadas mediante una relación de herencia forman una jerarquía de herencia.
La herencia es una técnica de abstracción que nos permite categorizar las clases de objetos bajo cierto criterio y nos ayuda a especificar las características de estas clases.
La palabra clave ***extends*** *define la relación de herencia.*
La frase ***«extends <SuperClase>»*** *especifica que esta clase es una subclase de la clase* ***SuperClase.***

public class <Nombre SubClase> extends <Nombre SuperClase>
{…}

La subclase define sólo aquellos campos que son únicos para los objetos de su tipo.
Los campos de la superclase se heredan y no necesitan ser incluidos en el código de la subclase.

Programación Concurrente -2023
Práctico Repaso del Paradigma Orientado a Objetos en JAVA

Práctica - Analice, pruebe y programe

Ejercicio 1:
Analice el siguiente código, indicando su funcionalidad:

```
import java.util.*;
class Instrumento {
   public void tocar() {
   System.out.println("Instrumento.tocar()");
}

public String tipo() {
  return "Instrumento";
}

public void afinar() {}
}
```

```
class Guitarra extends Instrumento {
   public void tocar() {
   System.out.println("Guitarra.tocar()");
}

public String tipo() { return "Guitarra"; }
   public void afinar() {}
}
```

```
class Piano extends Instrumento {
   public void tocar() {
   System.out.println("Piano.tocar()");
}

public String tipo() { return "Piano"; }
   public void afinar() {}
}
```

```
class Saxofon extends Instrumento {
   public void tocar() {
```

```
   System.out.println("Saxofon.tocar()");
}
public String tipo() { return "Saxofon"; }
   public void afinar() {}
}
```

```
// Un tipo de Guitarra
class Guzla extends Guitarra {
   public void tocar() {
   System.out.println("Guzla.tocar()");
}
public void afinar() {
   System.out.println("Guzla.afinar()");
}
}
```

```
// Un tipo de Guitarra
class Ukelele extends Guitarra {
     public void tocar() {
       System.out.println("Ukelele.tocar()");
}
public String tipo() { return "Ukelele"; }
}
```

```
public class Musica {
// No importa el tipo de Instrumento,
// seguirá funcionando debido a Polimorfismo:
    static void afinar(Instrumento i) {
    // ...
         i.tocar();
   }

   static void afinarTodo(Instrumento[] e) {
      for(int i = 0; i < e.length; i++)
              afinar(e[i]);
   }


    public static void main(String[] args) {
         Instrumento[] orquesta = new Instrumento[5];
         int i = 0;

        // Up-casting al asignarse el Arreglo
        orquesta[i++] = new Guitarra();
        orquesta[i++] = new Piano();
```

```
        orquesta[i++] = new Saxofon();
        orquesta[i++] = new Guzla();
        orquesta[i++] = new Ukelele();
        afinarTodo(orquesta);
}
   }  //clase Musica
```

Ejercicio 2:

Alquiler de Espacios para Aviones

En un aeropuerto se alquilan espacios de estacionamiento para aviones de distinto tipo. Para cada ALQUILER se guardan los datos del cliente, la fecha y hora de inicio de alquiler, la fecha y hora de finalización de alquiler, el número de la posición de estacionamiento y el avión que lo ocupará. Un AVIÓN se caracteriza por su matrícula, su envergadura en metros y el año de fabricación.

El cálculo del alquiler se realiza multiplicando la duración del alquiler en horas por un módulo que se obtiene multiplicando por 20 la envergadura en metros. Este valor se incrementa en una cantidad fija (250 en la actualidad). Es importante considerar que este valor puede cambiar en el futuro y puede variar en otros aeropuertos.

Ahora se desea diferenciar la información de algunos tipos de aviones:

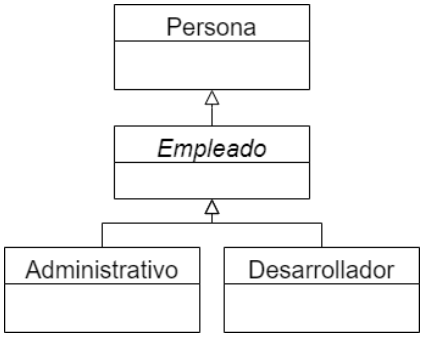
Número de motores para aviones a reacción.
Potencia en caballos de fuerza (HP) para aviones a hélice.
Potencia en HP y capacidad de pasajeros para aviones comerciales.
El módulo de los aviones de tipos especiales se obtiene como el módulo normal más:

El número de motores para aviones a reacción.
La potencia en HP para aviones a hélice.
La potencia en HP más la capacidad de pasajeros para aviones comerciales.

Diseñe el diagrama de clases, con detalle de atributos
Realice el diagrama de secuencia para el mensaje "calcularValor" de un alquiler. Utilice polimorfismo de forma apropiada. ¿Qué tipo de polimorfismo aplicó?
Realice el diagrama de secuencia para el mensaje "registrarAlquiler (posEstacionamiento, unBarco, unCliente, cantDias)" para registrar un nuevo alquiler en el puerto.

Programación Concurrente -2023
Práctico Repaso del Paradigma Orientado a Objetos en JAVA

Ejercicio 3:
Sistema de Empleados en una Empresa de Desarrollo de Software



En las empresas de desarrollo de software, se gestionan empleados de diferentes roles: Desarrolladores y Administrativos. Cada **persona** tiene información personal como nombre, DNI, dirección, fecha de nacimiento y sexo. Un **empleado** es una persona que desempeña una función en alguna Empresa y recibe un salario por ello. Todos los **empleados** tienen un legajo y cobran un porcentaje por antigüedad. Los **desarrolladores** son empleados que poseen habilidades técnicas y tienen un título que los avala. Los desarrolladores cobran un adicional basado en su rol. Los empleados **administrativos** reciben un adicional de acuerdo a su categoría y también tienen un adicional por asistencia.
Utilice la clase persona definida en la materia Programación Orientada a Objetos, implementada en Java, realizando sobre ella los cambios que crea necesarios.

a. Indique de cada clase, variables de instancias y métodos más representativos de la aplicación. Además considere una clase Empresa y grafíquela en el diagrama de clases.
b. Realice un diagrama de secuencia e implementación en Java para la siguiente interacción: "Generar una colección con los empleados con antigüedad mayor a 10 años"
c. Utilice polimorfismo para resolver el problema de mostrar por pantalla los datos de cada empleado. Utilice las utilidades del lenguaje y solo implemente lo estrictamente necesario. ¿Qué tipo de polimorfismo aplicó?
d. Considere que se posee una colección (cualquiera) con elementos que tienen la información correspondiente a "título (String)" y el "importe adicional que se cobra por el mismo". En qué clase la colocaría, que tipo de variable sería y qué tipo de método utilizaría para trabajar sobre ella?
e. Realice el diagrama de secuencia para el mensaje cobroMensualEmpleados() que genere una colección cuyos elementos mantengan la información referente al sueldo final que debe cobrar cada empleado. Implemente en Java todos los métodos necesarios
f. Resuelva la siguiente interacción: "Genere una colección ordenada por legajo de todos los empleados con título XX".
g. Ahora agregue la siguiente restricción: Cada empleado tiene almacenada la cantidad de horas trabajadas. Si esta es mayor que un máximo de 160 se cobra un plus del 5% del sueldo por horas extras.
h. Los empleados Administrativos además cobran un plus por presentismo, que es del 10% sobre el básico. En el caso de los técnicos, si la cantidad de horas trabajadas es mayor a 100 se cobra un valor fijo de 50$ por hora trabajada adicional hasta las 160 horas.

Con esta información modifique los métodos que crea convenientes para el cobro de un determinado mes.

Programación Concurrente -2023
Práctico Repaso del Paradigma Orientado a Objetos en JAVA

PREGUNTAS TEÓRICAS - Investigue, pruebe y conteste

Ejercicio 1:

Indicar V o F. Justificar

a. Los Constructores pueden llamar a otros constructores.
b. Un método abstracto en una clase no puede tener una implementación en la misma clase
c. Se usa this para invocar a un constructor dentro de la clase.
d. El constructor de una clase derivada no puede invocar al constructor de una clase base.
e. Si una clase redefine un método de la clase base, la versión en la clase derivada no afecta a la de la clase base.
f. Siempre se puede crear una instancia de una clase.
g. Una subclase puede tener métodos abstractos.
h. Las interfaces en Java solo pueden contener métodos abstractos.
i. La sobreescritura de métodos permite a una subclase proporcionar una implementación diferente para un método heredado de la clase base.

Ejercicio 2

Responda las siguientes preguntas.

1. Explique cómo se define una clase abstracta en JAVA.
2. ¿Cuál es la diferencia entre una clase abstracta y una interfaz en Java?
3. ¿Cómo se pueden declarar datos que son compartidos por todas las instancias de una clase dada?.
4. Indique cual es el efecto de la aplicación del modificador "final" en:
   a. una variable instancia.
   b. una variable local
   c. un método de instancia
   d. una clase

5. ¿Qué sucede cuando se tiene una variable final en blanco?
6. Analice el siguiente trozo de código e indique que sucede

```
package repaso;
public class PruebaStatic {
   public static String aCadena(){
         return "estoy en la superclase";
   }
```

```
}
package repaso;
public class RedefinicionStatic extends PruebaStatic {
   public static String aCadena(){
          return super.aCadena() + "estoy en la subclase" ;
   }
}
```

7. ¿Cuál es el alcance de una variable local de un método?
8. ¿Cuando se inicializa un parámetro? ¿Existe una inicialización por defecto?
9. ¿Qué significa hacer un "casting"? ¿Cuándo es necesario? ¿En Smalltalk se requiere hacer casting? ¿por qué?
10. ¿Qué sucede cuando declara un método como **protected** en una subclase que está en el mismo paquete que su superclase? ¿Y cuando la subclase está en otro paquete?
11. Analice el siguiente código:

```
public class Padre {
    public void hacerAlgo() {...}
}

public class Hijo extends Padre {
   private void hacerAlgo() {...}
}
```

```
public class Usa {
      public void hacerCosas(){
          Padre p = new Padre();
          Padre h = new Hijo();
          p.hacerAlgo();
          h.hacerAlgo();
      }
}
```

a. ¿Qué problema encuentra?
b. ¿Cómo lo resuelve?

12. Considere las clases PruebaUno y RedefinicionUno dadas a continuación:

```
package repaso;
   public class PruebaUno {
      protected int temp;
      public PruebaUno(int valor) {
          temp = valor;
      }
```

Programación Concurrente -2023
Práctico Repaso del Paradigma Orientado a Objetos en JAVA

```
   public void cambiar(int valor){
temp = valor;
   }
   public String aCadena(){
return "estoy en la superclase con valor " + temp;
   }
}
```

```
package repaso;
public class RedefinicionUno extends PruebaUno {
  private int temp2;
  public RedefinicionUno(int valor){
       super(valor);
       temp2 = valor * 7;
  }
  public String aCadena(){
      return "ahora estoy en la subclase con valor: " + temp ;
  }
  public String aCadena2() {
      return "en la subclase con valor: " + temp2;
   }
}
```

a. ¿Qué tipo de polimorfismo se utiliza en el método aCadena? ¿Cómo puede mejorarlo?
b. ¿Qué pasaría si el paquete de RedefinicionUno fuera prueba y no repaso?
c. Analice el siguiente código. Identifique errores y corríjalos.

```
package repaso;
   public class TestPruebas {
     public static void main(String[] args) {
PruebaUno aux[]= new RedefinicionUno[3];
aux[0]= new PruebaUno(4);
aux[1]= new RedefinicionUno(4);
aux[2]= new PruebaUno(14);
for ( PruebaUno elem: aux)
         System.out.println(elem.aCadena());
                              }
}
```

Considere ahora el siguiente código para el main. ¿Qué sucede? ¿Cómo lo resuelve? Justifique

```
public static void main(String[] args) {
         PruebaUno aux[]= new PruebaUnoUno[3];
         aux[0]= new PruebaUno(4);
         aux[1]= new RedefinicionUno(4);
```

```
          aux[2]= new PruebaUno(14);
          for ( PruebaUno elem: aux)
               System.out.println(elem.aCadena());
          for ( PruebaUno elem:aux)
               System.out.println(elem.aCadena2());
}
```