CESUR
Tu Centro Oficial de FP

EXAMEN PARCIAL – UNIDADES 1 Y 2		
TRIMESTRE: PRIMERO		**Fecha:**
CICLO: Desarrollo de Aplicaciones Web.	*CURSO: 1º*	*CALIFICACIÓN:*
MÓDULO: Programación	*Turno: Mañana*	
Nombre:	***Apellidos:***	

Instrucciones: *Esta prueba tiene como finalidad evaluar los aprendizajes de Programación. Lee atentamente y responde escribiendo el código más adecuado.*

Si las instrucciones no se siguen como se especifican el examen no será evaluado

PARTE PRÁCTICA. TIPO A.

- El examen práctico tiene una puntuación máxima de 10 puntos.
- Para superar la parte práctica se requiere alcanzar un mínimo de 5 puntos.

CESUR
Tu Centro Oficial de FP

1. **(3 puntos)** Diseña una aplicación Java que gestione distintos tipos de vehículos:

   Clase Base: Vehiculo

   Representa un vehículo genérico.

   - **Atributos:**
     - velocidad (entero): Velocidad promedio del vehículo.
   - **Métodos:**
     - **mover()**: Imprime un mensaje genérico indicando que el vehículo está en marcha.
     - **Sobrecarga de mover()**:
       - mover(int distancia): Muestra cuántos kilómetros recorrió el vehículo.
       - mover(int distancia, String terreno): Muestra cuántos kilómetros recorrió el vehículo y el tipo de terreno.
     - **toString()**: Devuelve una representación en texto de los atributos del vehículo.

   Subclase: Coche

   - **Atributos Adicionales:**
     - modelo (String): El modelo del coche (por ejemplo, "Sedán", "SUV").
   - **Métodos:**
     - Sobrescribe el método mover() para personalizar los mensajes específicos de un coche, indicando que el coche está en marcha.
     - Sobrescribe las sobrecargas de mover(int distancia) y mover(int distancia, String terreno) para incluir información del modelo.
     - Sobrescribe el método toString() para incluir el modelo del coche.

   ---

   3. Subclase: Bicicleta

   - **Atributos Adicionales:**
     - tipo (String): El tipo de bicicleta (por ejemplo, "Montaña", "Carretera").
   - **Métodos:**
     - Sobrescribe el método mover() para indicar que la bicicleta está en marcha.
     - Sobrescribe las sobrecargas de mover(int distancia) y mover(int distancia, String terreno) para incluir información del tipo de bicicleta.
     - Sobrescribe el método toString() para incluir el tipo de bicicleta.

   Generar constructores, getters y setters y el método toString().

Probar con este método main:

```
public static void main(String[] args) {
    Vehiculo vehiculo = new Vehiculo(60);
    Coche coche = new Coche(120, "Sedán");
    Bicicleta bicicleta = new Bicicleta(15, "Montaña");
    vehiculo.mover();
    coche.mover();
    bicicleta.mover();
    System.out.println();
    vehiculo.mover(50);
    coche.mover(120);
    bicicleta.mover(15);
    System.out.println();
    vehiculo.mover(100, "asfaltado");
    coche.mover(200, "carretera");
    bicicleta.mover(10, "terreno montañoso");
}
```

Ejemplo de ejecución:

```
/Users/jose/Library/Java/JavaVirtualMachines/openjdk-23.0.1/Contents,
El vehículo está en marcha.
El coche está en marcha.
La bicicleta está en marcha.

El vehículo recorrió 50 kilómetros.
El coche recorrió 120 kilómetros.
La bicicleta recorrió 15 kilómetros.

El vehículo recorrió 100 kilómetros en un terreno asfaltado.
El coche recorrió 200 kilómetros en un terreno carretera.
La bicicleta recorrió 10 kilómetros en un terreno terreno montañoso.

Process finished with exit code 0
```

2. **(3 puntos)** Desarrolla un programa en Java que simule un sistema de gestión de boletos para diferentes servicios, como trenes, vuelos y películas. Cada tipo de boleto debe tener reglas específicas para calcular su precio, y el sistema debe ser capaz de manejar precios dinámicos basados en condiciones como horarios pico. Además, se requiere un cálculo del costo total de todos los boletos adquiridos.

Clase Ticket

- **Atributos:**
  - ticketNumber (String): Identificador único del boleto.
  - basePrice (double): Precio base del boleto.
- **Métodos:**
  - **Constructor:** Inicializa el número del boleto y el precio base.
  - **Getters y Setters:** Métodos para acceder y modificar los atributos.
  - **calculatePrice(boolean isPeakTime)**:
    Método que calcula el precio del boleto. La implementación básica debe devolver el basePrice. Este método será sobrescrito en las subclases para aplicar reglas específicas.

Para el método **calculatePrice(boolean isPeakTime)** en cada una de las subclases:

TrainTicket

- **Reglas de cálculo de precio:**
  - Durante horarios pico (isPeakTime = true), se aplica un recargo del 10% al precio base.

FlightTicket

- **Reglas de cálculo de precio:**
  - Durante horarios pico (isPeakTime = true), se aplica un recargo del 20% al precio base.

MovieTicket

- **Reglas de cálculo de precio:**
  - Durante horarios pico (isPeakTime = true), se aplica un recargo fijo de 2€ sobre el precio base.

Generar constructores, getters y setters y el método toString().

Probar con este método main:

```
public static void main(String[] args) {
    TrainTicket trainTicket = new TrainTicket("T123", 100);
    FlightTicket flightTicket = new FlightTicket("F456", 200);
    MovieTicket movieTicket = new MovieTicket("M789", 15);
    boolean isPeakTime = true;
    double total = 0;
    double trainPrice = trainTicket.calculatePrice(isPeakTime);
    System.out.printf("Ticket %s - Precio: %.2f€\n",
                      trainTicket.getTicketNumber(), trainPrice);
    total += trainPrice;

    // Calcular precio del boleto de vuelo
    double flightPrice = flightTicket.calculatePrice(isPeakTime);
    System.out.printf("Ticket %s - Precio: %.2f€\n",
                      flightTicket.getTicketNumber(), flightPrice);
    total += flightPrice;

    // Calcular precio del boleto de cine
    double moviePrice = movieTicket.calculatePrice(isPeakTime);
    System.out.printf("Ticket %s - Precio: %.2f€\n",
                      movieTicket.getTicketNumber(), moviePrice);
    total += moviePrice;

    // Total de boletos
    System.out.printf("%nTotal de boletos: %.2f€\n", total);
}
```

Ejemplo de ejecución:

```
Ticket T123 - Precio: 110,00€
Ticket F456 - Precio: 240,00€
Ticket M789 - Precio: 17,00€

Total de boletos: 367,00€
```

3. **(4 puntos)** Diseña un sistema para gestionar animales en un zoológico.

   Clase base Animal:

   Atributos: nombre, edad.

   Métodos:

   - emitirSonido(): imprime un mensaje genérico como "El animal emite un sonido."
   - alimentar(): imprime "El animal está siendo alimentado."
   - Constructor para inicializar los atributos.

   Crea las siguientes subclases:

   Mamifero:

   Atributo adicional: tipoDeAlimentacion (carnívoro, herbívoro, omnívoro).

   Sobrescribe emitirSonido() con un mensaje genérico para mamíferos.

Ave:

Atributo adicional: puedeVolar (boolean).

Sobrescribe emitirSonido() con un mensaje genérico para aves.

Reptil:

Atributo adicional: esVenenoso (boolean).

Sobrescribe emitirSonido() con un mensaje genérico para reptiles.

Generar constructores, getters y setters y el método toString().

Probar con este método main:

```
public static void main(String[] args) {
    Mamifero leon = new Mamifero("León", 5, "carnívoro");
    Ave loro = new Ave("Loro", 2, true);
    Reptil serpiente = new Reptil("Serpiente", 3, true);
    leon.mostrarAlimentacion();
    leon.emitirSonido();
    leon.alimentar();
    loro.mostrarHabilidadVuelo();
    loro.emitirSonido();
    loro.alimentar();
    serpiente.mostrarPeligrosidad();
    serpiente.emitirSonido();
    serpiente.alimentar();
}
```

Ejemplo de ejecución:

```
El mamífero León es carnívoro.
El mamífero León emite un sonido específico.
El animal está siendo alimentado.
El ave Loro puede volar.
El ave Loro canta.
El animal está siendo alimentado.
El reptil Serpiente es venenoso.
El reptil Serpiente emite un sonido bajo.
El animal está siendo alimentado.
```