Primer Certamen

En este certamen usted no podrá hacer preguntas. Si algo no está claro, indíquelo en su respuesta, haga una suposición razonable y resuelva conforme a ella.

Primera parte, **sin apuntes** (32 minutos; 32 puntos):

1.- Responda brevemente y entregue esta hoja con su nombre. (Cuide su caligrafía, 4 puntos cada respuesta)

a)

```
class Persona {
   public void m() {
     System.out.println("P");
   }
}
class Estudiante extends Persona {
  public final void m() {
    System.out.println("E");
    super.m();
  }
}
class Sansano extends Estudiante {
  public void k() {
    m();
  }
}
```

Si alguna de las siguientes líneas de código genera error de compilación, indique cuál.

```
Persona p = new Estudiante();
p.m();
p=new Sansano();
p.m();
Estudiante e = new Sansano();
e.m();
Sansano s = (Sansano) e;
s.k();
```

Luego de comentar llamados con error, indique qué imprime el código restante.

* El código mostrado no presenta líneas con error.
* Salida (Cada invocación muestra lo mismo E y luego P):

E
P
E
P
E
P
E
P

b) ¿Qué diferencia existe entre el operador == y el uso del método equals(…)?
Un estudiante revisa una clase y no encuentra en ella la implementación del método equals(…) ¿por qué no se produce error al invocarlo sobre una instancia de esa clase?

Diferencia entre == y método equals(…): El primero compara dos referencias, será verdadero si dos nombres (o referencias a objetos) se refieren al mismo objeto. El método equals(…) compara dos objetos por igualdad.

No se produce error pues todo objeto hereda de la clase Object aunque no lo explicite. La clase Object implementa el método equals(…) por ello no hay error al invocarlo.

c) En clases vimos que podemos decir "un cuadrado es un rectángulo de lados iguales". Mencione otro ejemplo de dos clases donde esté presente la relación es-un pero no se cumpla el principio de sustitución. Explique por qué no se cumple en principio de substitución.

Múltiples respuestas son válidas aquí.
Alguien pudo decir: un punto **es un** círculo de radio cero.

Hay relación es-un pero no se cumple el principio de substitución pues si una instancia de esta clase Punto acude por sustitución en lugar de un círculo, cuando el usuario de círculo cambie su radio, el punto no podrá hacerlo. Así se demuestra que un círculo no puede ser sustituido por una instancia de esa clase Punto.

d) Los teléfonos celulares pueden hacer y recibir llamadas telefónicas. Podemos decir que un teléfono celular **es un** teléfono. Como se pueden tomar fotos, podemos también decir que un teléfono celular **es una** cámaras fotográficas. Considerando estos comportamientos ¿Qué declaraciones pondría usted en los archivos Telefono.java, Camara.java, y TelefonoCelular.java, ? Omita los métodos y atributos de cada clase. Interesa la primera línea de cada clase.

```
public class TelefonoCelular extends Telefono implements Carama { ... }
public interface Camara {....}
public class Telefono {…. }
```

e) Considere la siguiente clase:

```
class A {
        private int x,y;

        public A(int x, int y){
                this.x = x;
                this.y = y;
        }
}
```

Modifíquela para que permita realizar una copia profunda de una instancia de A.

Basta con implementar la interfaz Cloneable y el método clone. Al ser los atributos x e y de tipos primitivos, **NO** es necesario hacer copia profunda.

```
class A implements Cloneable{
        private int x,y;

        public A(int x, int y){
                this.x = x;
                this.y = y;
        }

        public Object clone(){
                A a = (A) super.clone();
                return a;
        }
}
```

f) Considere el código de la siguiente interfaz.

```
public interface Persona{
        public void correr();
        public void saltar();
}
```

Indique si es posible utilizar esta interfaz en una expresión lambda. Si es posible, dé un ejemplo. En caso contrario, justifique.

No es posible. Para que se pueda utilizar una expresión lambda sobre una interfaz, es necesario que esta defina **un solo método**. Interfaces con dos o más métodos no pueden ser utilizados a través de expresiones lambda.

g) Considere el siguiente bloque de código try-catch

```
...
try{
        f(); // Genera Excepcion1
        g(); // Genera Excepcion2
} catch(Excepcion1 e1){
        System.out.println(“Excepcion 1”);
} catch(Excepcion2 e2){
        System.out.println(“Excepcion 2”);
}
...
```

Indique y justifique las salidas por pantalla cuando:

- Al ejecutar el código, solo el método f() producirá una excepción.
- Al ejecutar el código, solo el método g() producirá una excepción.
- Al ejecutar el código, ambos métodos f() y g() producirán una excepción.

- System.out.println(“Excepcion 1”);
- System.out.println(“Excepcion 2”);
- System.out.println(“Excepcion 1”);

En el último caso, cuando se produce una excepción, esta se lanza inmediatamente, por lo que el código de la función g(), a pesar de que producirá una excepción, **NUNCA** alcanza a ejecutarse. Por lo que solo se muestra el mensaje producido por la excepción de f().

h) Para el desarrollo de una aplicación Android, necesita acceder a la lista de contactos del usuario. Indique qué clase provista por Android debe utilizar para esta tarea y por qué.

Es necesario utilizar la clase **Intent**, la cual se encarga de indicarle a Android que se desea acceder a la lista de contactos. De esta manera, Android manejará internamente cuál es la aplicación que responderá a la petición.

Segunda Parte, con apuntes (68 minutos) **Responda en hojas separadas.**

2.- (34 puntos) Una empresa de videojuegos ha requerido sus servicios para completar su último juego de batallas por turnos llamado ***Battle Monsters.***
Cada personaje posee un ***nombre*** *(name),* ***tipo*** *(type)*, ***puntos de vida*** *(hp)* y ***puntos de ataque*** *(attk)*, características que son obtenidas de los archivos ***playerA.csv*** y ***playerB.csv***, los cuales son argumentos de entrada a la ejecución del programa.
Forma de batalla: En cada turno, ambos personajes obtenidos de los archivos de entrada, se enfrentan de forma **simultánea** (no importa el orden)**.** El enfrentamiento produce que los puntos de ataque de un personaje se **resten** de los puntos de salud del adversario y viceversa, **hasta que alguno de los dos tenga 0 o menos puntos de salud**. Los puntos de ataque son **modificados por multiplicadores** dependiendo del **tipo** de los personajes según aparecen en la tabla 1. Los resultados posibles son **WinnerA** (gana jugador A), **WinnerB** (gana jugador B) o **Tie** (empate).

		Tipos (defensor)	
		Fire (0)	**Water (1)**
Tipos (atacante)	**Fire (0)**	x 1	x 0.5
	Water (1)	x 2	x 1

Tabla 1. Multiplicadores de ataque dependiendo del tipo del atacante y del defensor. **Ej**. Si el atacante es de tipo **Fire** y el defensor es de tipo **Water**, los puntos de ataque del atacante se multiplican por 0.5 (x 0.5).

Ejemplo de ejecución del programa:
$java BattleMonsterGame playerA.csv playerB.csv
¡Batalla! Foxy vs Rowdy
¡Ganador Foxy!

a) Complete el código de la clase ***Monster*** con su constructor y los métodos ***getHP()***, ***getAttk()*** y ***getName()***.
b) Complete los códigos de las clases ***FireMonster*** y ***WaterMonster***.
c) Complete el método ***battleMonsters(Monster mA, Monster mB)*** de la clase ***BattleMonsterGame***, el cual se encarga de enfrentar a dos monstruos ***mA*** y ***mB*** y retorna el resultado de la batalla.

OJO: ¡Solo deben completar los códigos solicitados!. Descargue los códigos de ayuda en la página de aula.

Monster.java

```
/**
 * Clase con las definiciones generales para cada monstruo
 */
public abstract class Monster {
   protected String name;
   protected int hp;
   protected int attk;

   /**
    * Constructor para inicializar atributos internos
    * @param name Nombre del Monster
    * @param hp Puntos de salud del Monster
    * @param attk Puntos de ataque del Monster
    */
   public Monster(String name, int hp, int attk){
      this.name = name;
```

```
        this.hp = hp;
        this.attk = attk;
    }

    /**
     * Metodo para obtener la cantidad de puntos de salud
     * @return Puntos de salud del Monster
     */
    public int getHP(){
        return hp;
    }

    /**
     * Metodo para obtener la cantidad de puntos de ataque
     * @return Puntos de ataque del Monster
     */
    public int getAttk(){
        return attk;
    }

    /**
     * Metodo para obtener el nombre
     * @return Nombre del Monster
     */
    public String getName(){
        return name;
    }

    /**
     * Método abstracto encargado de recibir el ataque producido por un Monster m.
     * Se implementa en las clases hijas.
     * @param m Monster atacante
     */
    public abstract void attackedBy(Monster m);
}
```

b)

FireMonster.java

```
/**
 * Clase FireMonster que hereda de Monster para MonsterType Fire
 */
public class FireMonster extends Monster {
    public FireMonster(String name, int hp, int atk) {
        super(name,hp,atk);
    }

    /**
     * Implementacion del metodo attackedBy para MonsterType Fire
     * @param m Monster atacante
     */
    public void attackedBy(Monster m){
        if(m instanceof WaterMonster){
            super.hp -= 2*m.getAttk();
        }else {
            super.hp -= m.getAttk();
        }
    }
```

WaterMonster.java

```
/**
 * Clase WaterMonster que hereda de Monster para MonsterType Water
 */
public class WaterMonster extends Monster{
    public WaterMonster(String name, int hp, int atk) {
        super(name,hp,atk);
    }

    /**
     * Implementacion del metodo attackedBy para MonsterType Water
     * @param m Monster atacante
     */

    public  void attackedBy(Monster m){
        if(m instanceof FireMonster){
            super.hp -= 0.5*m.getAttk();
        }else{
            super.hp -= m.getAttk();
        }
    }
}
```

c)

Método battleMonster

```
/**
 * Metodo encargado de enfrentar dos Monster
 * @param mA Monster A
 * @param mB Monster B
 * @return  resultado de la batalla:
 *         BattleResult.WinnerA, BattleResult.WinnerB o BattleResult.Tie
 */
public static BattleResult battleMonsters(Monster mA, Monster mB) {
    System.out.print("¡Batalla! ");
    System.out.println(mA.getName() + " vs " + mB.getName());
    while(mA.getHP()>0 && mB.getHP()>0){
        mA.attackedBy(mB);
        mB.attackedBy(mA);
    }
    if(mA.getHP() > 0){
        return BattleResult.WinnerA;
    }else if(mB.getHP()>0){
        return BattleResult.WinnerB;
    }else{
        return BattleResult.Tie;
    }
}
```

3.- (34 Puntos) Pregunta inspirada en Tareas: Auto controlado por teclado.
Se pide crear un auto que pueda moverse en el plano al presionar las letras i,j, y k. Al presionar “i” el auto se mueve 10 unidades hacia adelante. Al presionar “j” el auto gira hacia la izquierda en 45°, sin avanzar. Al presionar k el auto gira a la derecha en 45° y tampoco avanza. Este auto no tiene marcha atrás.
Para cada parte entregue sus **respuestas en directorios P3a.zip, P3b.zip y P3c.zip con los códigos de**

cada respuesta (si lo desea, puede ser .tar)

a) (16 puntos) Entregue el archivo Car.java para la descripción del auto y el archivo DrivenCar.java para la aplicación java que muestra la posición y dirección del auto cada vez que el usuario presiona alguna de las teclas i,j, y k. El formato de salida es : <posición x>, <posición y>, <águlo>
DrivenCar termina cuando el usuario presiona la letra e.
Suponga que el usuario no presiona otras teclas fuera de i,j,k, y e.

Car.java:

```
public class Car {
   public Car(double x, double y) {
      this.x=x;
      this.y=y;
   }
   public void turnLeft(){
      angle-=Math.PI/4;
   }
   public void turnRight(){
      angle+=Math.PI/4;
   }
   public void move(){
      x+=10*Math.cos(angle);
      y+=10*Math.sin(angle);
   }
   public String toString() {
      return x + ", " + y + ", " + angle + "\n";
   }
   private double x, y;
   private double angle;
}
```

DrivenCar.java

```
import java.io.IOException;
import java.io.InputStream;
public class DrivenCar {
   public static void main(String[] argv) throws IOException {
      InputStream in = System.in;
      Car a = new Car(100, 150);
      int c;
      do {
         c=in.read();
         switch(c) {
         case 'i': a.move(); break;
         case 'j': a.turnLeft();break;
         case 'k': a.turnRight();break;
         default: System.out.println(a);
         }
      }while (c !='e');
   }
}
```

b) (12 puntos) Cree el archivo CarView.java. La vista del auto será solo un rectángulo en la pantalla de 40x30 pixeles.

Entregue los archivos Car.java, CarView.java y GraphicDrivenCar.java para operar el auto con las letras i, j, y k y que éste muestre su vista en una interfaz gráfica JavaFX en la posición y dirección correspondiente. Es decir, el rectángulo se mueve y gira acorde a las letras i,j, y k presionadas.

Car.java:

```
public class Car {
   public Car(double x, double y) {
      this.x=x;
      this.y=y;
      view = new CarView(this);
   }
   public void turnLeft(){
      angle-=Math.PI/4;
      view.update();
   }
   public void turnRight(){
      angle+=Math.PI/4;
      view.update();
   }
   public void move(){
      x+=10*Math.cos(angle);
      y+=10*Math.sin(angle);
      view.update();
   }
   public String toString() {
      return x + ", " + y + ", "+ angle + "\n";
   }
   public double getX(){
      return x;
   }
   public double getY(){
      return y;
   }
   public double getDir(){
      return angle;
   }
   public CarView getView() {
      return view;
   }
   private double x, y;
   private CarView view;
   private double angle;
}
```

CarView.java:

```
import javafx.scene.shape.Rectangle;
public class CarView extends Rectangle{
   public CarView (Car c) {
      super(c.getX(), c.getY(), 40, 30);
      car=c;
   }
   public void update(){
      setX(car.getX());
```

```
      setY(car.getY());
      setRotate(car.getDir()*180/Math.PI);
   }
   private Car car;
}
```

GraphicDrivenCar.java:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.Pane;
import javafx.scene.input.KeyEvent;

public class GraphicDrivenCar extends Application {
   public void start(Stage primaryStage) {
      Car car = new Car(100, 150);
      Pane pane = new Pane();
      pane.getChildren().add(car.getView());
      Scene scene = new Scene(pane, 300,300);
      scene.setOnKeyPressed(e->handleKey(e, car));
      primaryStage.setScene(scene);
      primaryStage.show();
   }
   private void handleKey(KeyEvent e, Car car){
      switch (e.getCode()) {
         case I: car.move(); break;
         case J: car.turnLeft(); break;
         case K: car.turnRight(); break;
         case E: System.exit(0); break;
      }
   }
}
```

c) (6 puntos) Modifique lo necesario para que cada instancia de Car emita un sonido mientras corre el programa.
Puede usar el sonido en "http://profesores.elo.utfsm.cl/~agv/elo329/1s20/C1/carSound.mp3"

Basta con modificar el archivo CarView.java:

```
import javafx.scene.shape.Rectangle;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;

public class CarView extends Rectangle{
   public CarView (Car c) {
      super(c.getX(), c.getY(), 40, 30);
      car=c;
      Media media = new Media(MEDIA_URL);
      MediaPlayer mediaPlayer = new MediaPlayer(media);
      mediaPlayer.setCycleCount(MediaPlayer.INDEFINITE);
      mediaPlayer.setVolume(0.1);
      mediaPlayer.play();
   }
   public void update(){
```

```
        setX(car.getX());
        setY(car.getY());
        setRotate(car.getDir()*180/Math.PI);
    }
    private Car car;
    private static final String MEDIA_URL =
        "http://profesores.elo.utfsm.cl/~agv/elo329/1s20/C1/carSound.mp3";
}
```