Primer Certamen

Primera Parte (Sin Apuntes, 30 minutos, 30 puntos):

5 pts c/u el total se multiplica por 30/40 y redondea.

1. ¿Qué significa polimorfismo? ¿Cuáles son los distintos tipos de polimorfismo?

   Polimorfismo es la cualidad de usar un mismo nombre para referirse a cosas similares.
   Algunos tipos de polimorfismo son:
   a) Cuando vía herencia un mismo nombre, instancia de la clase hoja, puede tomar también el lugar de una instancia de la clase base.
   b) Hay polimorfismo cuando varias clases implementan una interfaz. Luego podemos crear código genérico en el sentido que depende sólo de los métodos de la Interfaz. Este código genérico puede ser usado con objetos de distintas clases que implementan la interfaz.
   c) Hay polimorfismo cuando en una clase base definimos métodos abstractos que luego las clases derivadas definen. Este es un caso similar al b).
   d) Algunos autores también consideran polimorfismo cuando un mismo método puede ser invocado usando diferentes tipos de datos como parámetros (sobrecarga de métodos), pero otros no lo consideran así.
   Con un par de casos es OK.

2. Dé ejemplos simples de dos distintos tipos de polimorfismo.
   Primer ejemplo: Polimorfismo cuando hay herencia.

```
class Persona {
….
}
class Empleado extends Persona {
…
}
```

   Segundo caso: Polimorfismo cuando tenemos interfaces.

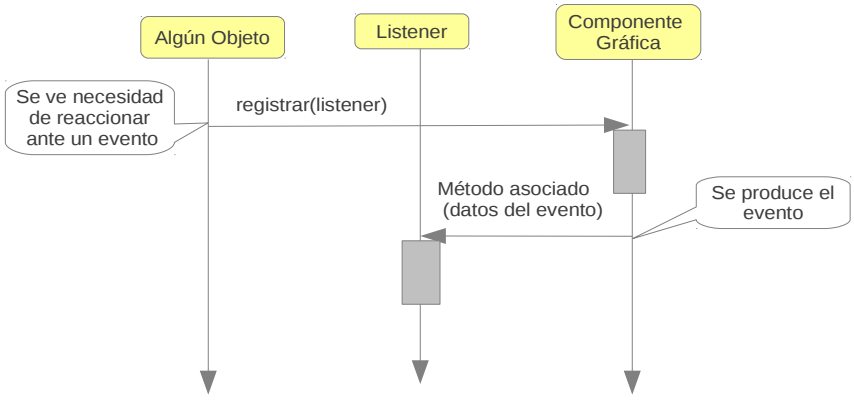
```
Class Empleado implements Comparable{
….
}
Empleado[] funcionarios = new Empleado[50];
/* Crear los empleados */
Array.sort(funcionarios);
```

3. ¿Qué es ligado dinámico? Indique un beneficio y un perjuicio de usar esta técnica.
   Ligado dinámico es la definición en tiempo de ejecución del código a ejecutar ante una invocación de un método.
   Beneficio: permite crear código reutilizable.
   Perjuicio: El ligado dinámico es más lento de ejecutar que el estático definido en tiempo de compilación.

4. ¿Para qué sirve la clase Class? De un ejemplo simple de su uso.

La clase Class permite consultar información sobre la clase de un objeto en tiempo de ejecución.
class Source {
..
}
System.out.println((new Source()).getClass.getName());
Retorna Source.

5. ¿En qué contexto en Java se usan los listeners? Haga un diagrama de secuencia para mostrar los pasos generales para su uso.
   Los listener se usan para registrar código a ejecutar ante la ocurrencia de un evento.



6. ¿Describa por qué motivo se extiende JFrame en el contexto de los GUIs de Java? ¿Qué parte juegan los listeners en esta implementación?
   Se extiende JFrame porque ésta es una componente gráfica de nivel superior; es decir, puede existir por sí sola en un escritorio.
   Los listener llevan a cabo la ejecución de la lógica del programa cada vez que un evento ocurre.
   Otra respuesta es: los listener permiten implementar la programación conducida por eventos.

7. ¿Qué beneficios otorgan las excepciones? ¿Cuántos tipos hay y cuales son?
   El beneficio de las excepciones es separar las condiciones de error de la lógica principal de un segmento de código (o método).
   Hay de dos tipos, aquellas generadas por el lenguaje; por ejemplo, cuando se accede a una referencia null. El otro tipo son las incluidas por el programador.

8. ¿Las excepciones siempre tienen que ser capturadas? ¿Por qué? ¿En qué caso no es necesario capturar una excepción? ¿En qué caso no es posible relanzar una excepción?
   No, por que una excepción también puede ser relanzada.
   No es necesario capturar excepciones cuando son generadas por el lenguaje (división por cero, etc).
   No es posible relanzar una excepción cuando estamos redefiniendo un método que no lo hace.

Segunda Parte (Con Apuntes, 60 minutos, 70 puntos):

9. Para este código utilice instance of y Casteo para lograr que el arreglo staff almacene Managers y Employees. También debe poder almacenar valores de bonus en staff de tipo Manager, debe poder imprimir toda la información de staff (incluyendo el bono de los Managers). Debe modificar las clases y métodos si es necesario. (25 puntos)

```
import java.util.*;
public class Main {
   public static void main(String[] args) {
      Employee[] staff = new Employee[3];
      // fill the staff array with Manager and Employee objects
      staff[0] =
      staff[1] =
      staff[2] =
      // using cast set bonus information for Manager staff to 9999
      for (int i = 0; i < staff.length; i++) {
        ...
      }
      // print out ALL information about all staff including Managers and Employees
      for (int i = 0; i < staff.length; i++) {
        ...
      }
   }
}

class Employee {
   public Employee(String n, double s,
         int year, int month, int day) {
      name = n;
      salary = s;
      GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
      // GregorianCalendar uses 0 for January
      hireDay = calendar.getTime();
   }
    ...
}

class Manager extends Employee {
 public Manager(String n, double s,
         int year, int month, int day) {
      super(n, s, year, month, day);
      bonus = 0;
```

```
    }
     ...
   private double bonus;
}
```

Solución:

```
import java.util.*;
/**
 * @author tomas agustin
 */
public class Main {
   public static void main(String[] args) {
      Employee[] staff = new Employee[3];
      // fill the staff array with Manager and Employee objects
      staff[0] = new Manager("Carl Cracker", 80000, 1987, 12, 15);    4 pts
      staff[1] = new Employee("Harry Hacker", 50000, 1989, 10, 1);
      staff[2] = new Employee("Tommy Tester", 40000, 1990, 3, 15);

      // using cast set bonus information for Manager staff to 9999
      for (int i = 0; i < staff.length; i++) {
         if (staff[i] instanceof Manager) {        5 pts
            Manager m = (Manager) staff[i];
            m.setBonus(9999);
         }
      }
      // print out information about all staff
      for (int i = 0; i < staff.length; i++) {
            Employee e = staff[i];                8 pts.
            System.out.print("name=" + e.getName() + ",salary=" + e.getSalary());
            if (e instanceof Manager) {
               Manager m = (Manager) e;
               System.out.print(",bonus=" + m.getBonus());
            }
             System.out.println();
         }
      }
   }
}

class Employee {
   public Employee(String n, double s,
         int year, int month, int day) {
      name = n;
```

```
        salary = s;
        GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
        // GregorianCalendar uses 0 for January
        hireDay = calendar.getTime();
    }
    public String getName() {        4 pts  resto de clase
        return name;
    }
    public double getSalary() {
        return salary;
    }
    public Date getHireDay() {
        return hireDay;
    }
    public void raiseSalary(double byPercent) {
        double raise = salary * byPercent / 100;
        salary += raise;
    }
    private String name;
    private double salary;
    private Date hireDay;
}

class Manager extends Employee {
    public Manager(String n, double s,
            int year, int month, int day) {
        super(n, s, year, month, day);
        bonus = 0;
    }
    public double getSalary() {       4 pts resto de clase
        double baseSalary = super.getSalary();
        return baseSalary + bonus;
    }
    public void setBonus(double b) {
        bonus = b;
    }
    public double getBonus() {
        return bonus;
    }
    private double bonus;
}
```

10. Modifique el código anterior para tener una clase abstracta base que se llame Person. En main utilice un ArrayList<Person> persons = new ArrayList<Person>(); para contener los Employees y Managers. Replique la funcionalidad de main del problema 9 con el ArrayList, muestre los cambios del código. (25 puntos)

```
abstract class Person
{
...
}
class Employee extends Person
{
...
}
class Manager extends Person
{
…
}
```

Solución:

```
import java.util.*;
public class Main {
   public static void main(String[] args) {
      ArrayList<Person> persons = new ArrayList<Person>();
      // fill the persons list with Manager and Employee objects
      persons.add(new Manager("Carl Cracker", 80000, 1987, 12, 15));      4 pts.
      persons.add(new Employee("Harry Hacker", 50000, 1989, 10, 1));
      persons.add(new Employee("Tommy Tester", 40000, 1990, 3, 15));
      // using list set bonus information for Manager staff to 9999
      for (int i = 0; i < persons.size(); i++) {
         if (persons.get(i) instanceof Manager) {      3 pts.
            Manager m = (Manager) persons.get(i);
            m.setBonus(9999);
         }
      }
      for (int i = 0; i < persons.size(); i++) {
         persons.get(i).print();       6 pts.
      }
   }
}
abstract class Person {
   abstract void print();
}
class Employee extends Person {
```

```
   public Employee(String n, double s, int year, int month, int day) {
      name = n;
      salary = s;
      GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
      // GregorianCalendar uses 0 for January
      hireDay = calendar.getTime();
   }
   public String getName() {
      return name;
   }
   public double getSalary() {
      return salary;
   }
   public Date getHireDay() {
      return hireDay;
   }
   public void raiseSalary(double byPercent) {
      double raise = salary * byPercent / 100;
      salary += raise;
   }
   public void print() {       4 pts
      System.out.println("name=" + name + ",salary=" + salary);
   }
   private String name;     2 pts atributos
   private double salary;
   private Date hireDay;
}

class Manager extends Person {
   public Manager(String n, double s, int year, int month, int day) {
      bonus = 0;
      name = n;
      salary = s;
      GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
      // GregorianCalendar uses 0 for January
      hireDay = calendar.getTime();
   }
   public String getName() {
      return name;
   }
   public Date getHireDay() {
      return hireDay;
```

```
   }
   public void raiseSalary(double byPercent) {
      double raise = salary * byPercent / 100;
      salary += raise;
   }
   public double getSalary() {
      return salary + bonus;
   }
   public void setBonus(double b) {
      bonus = b;
   }
   public double getBonus() {
      return bonus;
   }
   public void print() {      4 pts
      System.out.println("name=" + name + ",salary=" + salary+ ",bonus=" + bonus);
   }
   private double bonus;    2 pts por atributos
   private String name;
   private double salary;
   private Date hireDay;
}
```

11. Muestre el código de MouseHandler que muestre rectángulos en la pantalla cuando el usuario aprieta el mouse. Asuma que tiene una clase llamada g2 que puede llamar para dibujar: (20 pts).

```
      g2.draw(new Rectangle2D.Double(x, y, rectWidth, rectHeight));

 private class MouseHandler extends MouseAdapter
   {
...
   }
```

Solución:

```
  /* aquí va todo lo previo correspondiente a la clase con despliegue gráfico*/

 private class MouseHandler extends MouseAdapter /* clase anidada, tiene acceso a atributos x e y.*/
   {
     public void mousePressed(MouseEvent event)      10 pts. por método
     {
       x = event.getPoint().getX();
       y = event.getPoint().getY();
       repaint();
     }
```

```
  }
  public void paintComponent(Graphics g)     6 pts. por método
  {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    g2.draw(new Rectangle2D.Double(x, y, rectWidth, rectHeight));
  }
/* aquí otros atributos de la clase con despliegue gráfico */
  private int x, y;  /* atributo necesario para pintar rectángulo. */  4 pts por 4 atributos.
  private  int rectWidth=30;
  private  int rectHeight=20;
}
```