Trabajo Práctico Nº 3

Tema: PROGRAMACION ORIENTADA A OBJETOS

1.Dadas la clase Auto (incompleta):

```
class Auto
{
   String marca;
   int modelo;
   int puertas;
   String patente;
   float precio;

  Auto(String marca, int modelo, int puertas)
  {
     this.marca=marca;
     this.modelo=modelo;
     this.puertas=puertas;
     precio=0;
     patente="no patentado";
  }

  String devolverMarca()
  {
      return marca;
  }
  void cambiarMarca(String marca)
  {
      this.marca=marca
   }
  int devolverModelo()  {... }
  void cambiarModelo(int modelo)  {... }
  int devolverPuertas()  {... }
  void cambiarPuertas(int puertas)  {... }
  float devolverPrecio()  {... }
  void cambiarPrecio(float precio)  {... }
  String devolverPatente()  {... }
  void cambiarPatente(String patente)  {... }
}
```

a) Completar la implementación de los métodos declarados y generar el archivo class.

b) ¿Cuál es la salida de Prueba?

```
class Prueba
{
  public static void main(String [] args)
  {
     Auto miAuto;
     miAuto = new Auto("FIAT", 2000, 3);                    ****
     System.out.println("Modelo: "+  miAuto.devolverModelo());
     System.out.println("Marca: " + miAuto.devolverMarca());
     System.out.println("Puertas: "+  miAuto.devolverPuertas());
     System.out.println("Precio: "  +miAuto.devolverPrecio());
     System.out.println("Patente: " + miAuto.devolverPatente());
  }
}
```

c) Luego de la línea marcada con ******** agregue el siguiente código e indique la salida de Prueba.

```
miAuto.cambiarPrecio(9500));
miAuto.cambiarPatente("SHY 585"));
```

d) Idem c.

```
miAuto.cambiarPrecio(7300));
miAuto.cambiarPatente("BRT 259"));
```

```
        miAuto.cambiarPuertas(5));
```

2. Implementar y probar la ejecución de los siguientes TDA:
   a) una pila de enteros
   b) una lista simple enlazada de nodos
   c) una cola de enteros

3. Implementar y probar la clase Punto:

```
class Punto
{
    Punto(int x, int y)
    {    this.x = x;
         this.y = y;  }
    int devolverX()
    {   return x;  }
    void cambiarX(int x)
    {  this.x = x; }
    int devolverY()
    {   return Y; }
    void cambiarY(int y)
    {  this.y = y; }
    void moverPunto(int desplazamiento, char sentido)
    {
     // sentido será 'I' (izquierda) 'D' (derecha) 'A' (arriba) 'B' (abajo)
    // si sentido es 'I''o 'D' se modifica x , sumar o restar desplazamiento
    // si sentido es 'A''o 'B' se modifica y , sumar o restar desplazamiento
    }
    void moverPunto(int d1, char s1, int d2, char s2)
    {
     // por ejemplo el punto se mueve hacia arriba y hacia a izquierda
    }
}
```

4. Implementar la clase Rectángulo con al menos tres constructores (sobrecarga) y los siguientes métodos:

```
Punto devolverSI( ){ ... }
Punto devolverSD( ){ ...}
Punto devolverII( ){ ... }
Punto devolverID( ){ ...}
moverRectangulo(Punto puntoRectangulo, Punto nuevoPunto)
{
  // determinar que punto del rectángulo es puntoRectangulo y lo traslada a nuevoPunto
  // realiza el desplazamiento de los demás puntos del rectángulo
  // puede usar métodos de la clase Punto
}
cambiarRectangulo(Punto puntoRectangulo, Punto otroPunto)
{
  // este método redimensiona el rectángulo (agranda o achica, hacia fuera o hacia adentro)
  // cuando un puntoRectangulo del rectángulo se lleva hasta otroPunto es necesario modificar
  // otro punto mas del rectángulo
}
int devolverSuperficie()
{
   // este método calcula y devuelve la superficie del rectángulo
}
int devolverPerimetro()
{
  // este método calcula y devuelve el perímetro del rectángulo
```

```
}
}
```

5. Dado el siguiente problema implementar las clases que den solución al mismo:
Un cajero automático permite a los clientes de los bancos realizar depósitos , extracciones, consultas de saldos y transferencias de dinero. No se pueden realizar depósitos de cantidades negativas, ni extracciones si el saldo de la cuenta es inferior a dicho monto. El cajero mantiene un conjunto de cuentas. Cada cuenta tiene un titular, número (único para cada cuenta) y saldo.

6. Dada la siguiente jerarquía de herencia:

```
class A {
  int atributoA;
  void metodoA() { } }

class B extends A {
  int atributoB;
  void metodoB() { } }

class C extends B {
  int atributoC;
  void metodoC() { } }
```

a) ¿qué mensajes se pueden enviar a un objeto de la clase A?
b) ¿qué mensajes se pueden enviar a un objeto de la clase B?
c) ¿qué mensajes se pueden enviar a un objeto de la clase C?
d) ¿qué atributos describen a un objeto de la clase A?
e) ¿qué atributos describen a un objeto de la clase B?
f) ¿qué atributos describen a un objeto de la clase C?

7. Dada la siguiente jerarquía de herencia:

```
class A {
  int atributoA=5;
  void metodoA() { System.out.println(" *** metodo A" ***); } }

class B extends A {
  int atributoB=2;
  void metodoA() { System.out.println(" *** método A redefinido en B" ***);
  void metodoB() { System.out.println(" *** método B" ***);} }

class C extends B {
  int atributoC=0;
  int atributoB=7;
  void metodoC() { System.out.println(" *** método C" ***);} }
```

a) ¿qué ocurre cuando se le envía el mensaje metodoA a un objeto de la clase A?
b) ¿qué ocurre cuando se le envía el mensaje metodoA a un objeto de la clase B?
c) ¿qué ocurre cuando se le envía el mensaje metodoA a un objeto de la clase C?
d) ¿qué valor tiene la propiedad atributoB de un objeto de la clase B?
e) ¿qué valor tiene la propiedad atributoB de un objeto de la clase C?
f) ¿qué valor tiene la propiedad atributoB de un objeto de la clase A?
g) ¿qué valor tiene la propiedad atributoA de un objeto de la clase C?
h) ¿qué valor tiene la propiedad atributoA de un objeto de la clase B?

8. Dadas las siguientes jerarquías de herencia indique la salida que produce Prueba y sus conclusiones:

```
class Prueba
```

```
{
  public static void main (String [] args)
   {
      Hija x;
      x = new Hija();
      System.out.println("Atributo j de X:" + x.j);
      System.out.println("Atributo i de X:" + x.i);
   }
}
```

8.1.- *// clase base y clase derivada tienen constructores*

```
class Base {
  int i;
  Base()
    {  i =100; } }

 class Hija extends Base {
    int j;
    Hija()
    {  j =120;} }
```

8.2.- *// clase derivada no tiene constructor*

```
class Base {
  int i;
  Base()
    {  i =100; } }

 class Hija extends Base {
    int j; }
```

8.3.- *// clase base no tiene constructor sin argumentos*

```
class Base {
  int i;
  Base(int n)
    {  i =n; } }

 class Hija extends Base {
    int j;
    Hija()
    {  j =120;}  }
```

8.4.-

```
class Base {
  private int i;
  Base(int n)
    {  i =n; } }

 class Hija extends Base {
    int j;
    Hija()
    {  super (110);
       j =120;}  }
```

9. Una pizarra es un objeto en el cual se pueden añadir, eliminar y dibujar figuras. Las figuras con las que se puede trabajar son Triángulo (definido por los tres puntos vértices), rectángulo (definido por los cuatro puntos) y círculo (definido por punto central y radio). Una figura puede crearse y dibujarse. La pizarra tiene

una colección de 0 a n figuras. Aplicando los conceptos de herencia, agregación y polimorfismo realice el ejercicio.
Notas:

1. Cuando una triángulo se dibuja simplemente se visualiza el mensaje ”dibujando triángulo”, ídem para el resto de las figuras.
2. Puede utilizar la clase Vector del API como contenedor de figuras.

10. Dado el siguiente método de la clase X en java

```
void metodo (lista de parámetros)
   {
      --
      x   ¬
      --
   }
```

En un punto se hace referencia al identificador x, en que orden buscará el binding de la misma. (ambientes de referencia).

11. Explicar el uso y efecto de los siguientes operadores y/o enunciados de Java:
    a) una clase definida como final
    b) un método definido como final
    c) un atributo definido como final
    d) un método definido como static
    e) un atributo definido como static
    f) un método definido como private
    g) un atributo definido como private
    h) un método definido como protected
    i) un atributo definido como protected
    j) un método definido como public
    k) un atributo definido como public