**Practico 4**
**Diagramas de secuencia y comunicación**

---

*Ejercicios oblicagorios: 2,3,4*
*Entregar ejercicio 5*

---

1) A partir del siguiente código Java, construir el modelo de diseño dinámico a través de un diagrama de secuencia a partir de la invocación del mensaje “htmlStatement” a un objeto de la clase “Customer”.

```
package ar.edu.unsl.tuw.ingsw;
import java.util.Enumeration;
import java.util.Vector;

public abstract class Price {
	abstract int getPriceCode();
	abstract double getCharge(int daysRented);
	int getFrequentRenterPoints(int daysRented) {return 1;}
}
class Movie {
	private String _title;
	private Price _price;
	public Movie(String name, Price price) {
		_title = name;
		_price = price;
	}
	int getFrequentRenterPoints(int daysRented) {
		return _price.getFrequentRenterPoints(daysRented);
	}
	double getCharge(int daysRented) {return _price.getCharge(daysRented);}
	String getTitle() {return _title;}
}
class Rental {
	private Movie _movie;
	private int _daysRented;
	public Rental(Movie movie, int daysRented) {
		_movie = movie;
		_daysRented = daysRented;
	}
	public int getDaysRented() {return _daysRented;}
	public Movie getMovie() {return _movie;}
	public double getCharge() {return _movie.getCharge(_daysRented);}
	public int getFrequentRenterPoints() {
				return
			_movie.getFrequentRenterPoints(_daysRented);}
	}
class Customer {
	private String _name;
	private Vector _rentals = new Vector();
	public Customer(String name) {
		_name = name;
	}
	public void addRental(Rental arg) {_rentals.addElement(arg);}
	public String getName() {return _name;}
```

```
      public String htmlStatement() {
            Enumeration rentals = _rentals.elements();
            String result =
                  "<H1> Rentals for >EM>" + getName() + "</EM></H1><P>\n";
            while (rentals.hasMoreElements()) {
                  Rental each = (Rental) rentals.nextElement();
                  result += each.getMovie().getTitle() + ": "
                              + String.valueOf(each.getCharge()) + "<BR>\n";
            }
            return result;
      }
      public double getTotalCharge() { ... }
      public int getTotalFrequentRenterPoints() { . . . . . }}
}
class ChildrensPrice extends Price {
      @Override
      int getPriceCode() {...}
      @Override
      double getCharge(int daysRented) {...}
}
class NewReleasePrice extends Price {
      double getCharge(int daysRented) {return daysRented * 3;}
      int getFrequentRenterPoints(int daysRented) {return daysRented;}
      @Override
      int getPriceCode() {return 0;}
}
class RegularPrice extends Price {
      double getCharge(int daysRented) {return daysRented;}
      @Override
      int getPriceCode() {return 0;}
}
```

2) Construir el modelo de diseño dinámico a través de un diagrama de secuencia a partir de la invocación del mensaje “allPayToEmployeeTypes”.

```
     public abstract class EmployeeType {
           abstract int payAmount();
           abstract int getMonthlySalary();
     }
     public class Employee {
           private Set<EmployeeType> _employeeTypes = new HashSet<EmployeeType>();
           public Set<EmployeeType> getEmployeeTypes() {
                 return _employeeTypes;
           }
           public void addEmployeeTypes(EmployeeType c) {
                 _employeeTypes.add(c);
           }
           public void removeEmployeeTypes(EmployeeType c) {
                 _employeeTypes.remove(c);
           }
           public int allPayToEmployeeTypes() {
                 Set<EmployeeType> s = getEmployeeTypes();
                 Iterator<EmployeeType> iter = s.iterator();
                 int total = 0;
                 while (iter.hasNext()) {
                       EmployeeType each = (EmployeeType) iter.next();
                       total += each.payAmount();
```

```
            }
            return total;
      }
}
public class Engineer extends EmployeeType { @Override
      int getMonthlySalary() {
            return 0;
      }
      @Override
      int payAmount() {
            return this.getMonthlySalary();
      }
}
public class Manager extends EmployeeType {
      int getBonus(){
            return 0;
      }
      @Override
      int payAmount() {
            return this.getMonthlySalary() + this.getBonus();
      }
      @Override
      int getMonthlySalary() {
            return 0;
      }
}
public class Salesman extends EmployeeType{
      int getCommission(){
            return 0;
      }
      int payAmount() {
            return this.getMonthlySalary() + this.getCommission();
      }
      @Override
      int getMonthlySalary() {
            return 0;
      }
}
```

2) Construir el modelo de diseño dinámico a través de un diagrama de comunicación a partir de la ejecución del método “main” de la clase “App”.

```
public class App {
      private static ClaseA a;
      public static void main(String[]args) {
       a = new ClaseA();
       a.am1();
       a.am2();
      }
}
public class ClaseA {
      private ClaseB b;
      private int c;
      public void am1() {
            c = 1;
            b = new ClaseB();
            b.bm1(this);
```

```
                show();
        }
        public void am2() { }
        public void show() { System.out.println(c); }
}
public class ClaseB {

        private ClaseA a;

        public void bm1(ClaseA unA) {
                a = new ClaseA();
                a.show();
                unA.show();
                a = unA;
                a.show();
                unA.show();
        }
}
```

4) Defina los casos de uso para la realidad "Casa de Seguros" provista por la cátedra. Luego realice los diagramas de secuencia correspondientes a los flujos principales de los mismos.

5) Defina los diagramas de comunicación para el 30% de los casos de uso del Trabajo Especial.