Lenguajes de Programación
Soluciones a pruebas de nivel

Pedro García López

pgarcia@etse.urv.es/

Grid Computing

Se trata de modelar en un lenguaje orientado a objetos (Java) el problema de computación de recursos en una red de ordenadores. Un sistema Grid está formado por una lista de tareas y una lista de procesadores capaces de resolver tareas. De cada procesador del grid se sabe su identificador, IP, capacidad de proceso (GHz), memoria RAM (MB) y disco duro (GB).Para simplificar, supongamos que un procesador solo puede procesar una tarea en cada iteración, por lo que si tiene asignada una tarea estará ocupado y si no estará libre.

Una tarea tiene un identificador, una lista de valores de entrada (enteros, double, ...), un resultado de la computación, y la operación que se ha de aplicar sobre la lista de entrada para producir el resultado. Por ejemplo, si nos pasan una lista de enteros (4,2,3) y la operación es el sumatorio, el resultado será (9). Si la operación fuese el producto el resultado será 24. Un objetivo claro de la práctica es que el sistema sea facilmente **extensible** y se puedan incorporar nuevas **tareas** con nuevas operaciones **sin modificar el código**.

Sobre el Grid podremos añadir nuevos procesadores, eliminar a algun procesador, obtener la lista de procesadores, así como añadir Tareas, eliminar Tareas, y listar Tareas. El Grid ademas ofrecerá una función de asignar tareaS a procesadores (una tarea por procesador) que esten libres. Por último el Grid tendrá una operación de resolver tareas, que hará que cada procesador resuelva su tarea asignada y cambie su estado a disponible.

Crear un programa con un main que pruebe la funcionalidad del Grid **(crear grid, crear procesadores, crear tareas, añadir tareas y procesadores al grid,asignar tareas, resolver tareas y ver resultados).**

Solución básica 1

```java
import java.util.*;

public class Grid {
    private List procesadores;
    private List tareas;

    public Grid(List procesadores, List tareas) {
        this.procesadores = procesadores;
        this.tareas = tareas;
    }
    public Grid(){
        this.procesadores = new LinkedList();
        this.tareas = new LinkedList();
    }

    public void addTarea(Tarea nuevaTarea){
        tareas.add(nuevaTarea);
    }

    public void addProcesador(Procesador proc){
        procesadores.add(proc);
    }
    public List getTareas() {
        return tareas;
    }

    public void resolverTareas(){
        Iterator it = procesadores.iterator();
        while (it.hasNext()){
            Procesador proc = (Procesador)it.next();
            proc.resolverTarea();

        }

    }
```

```java
public class Procesador {

    private Tarea tarea;

    public Tarea getTarea() {
        return tarea;
    }

    public void setTarea(Tarea tarea) {
        this.tarea = tarea;
    }

    public void resolverTarea(){
        tarea.operacion();
    }
}
```

```java
public abstract class Tarea {
    protected int valores[];
    protected int result;
    private String identificador;


    public Tarea(int[] valores, String identificador) {
        this.valores = valores;
        this.identificador = identificador;
    }

    public abstract void operacion();

    public int getResult() {
        return result;
    }

    public String toString(){
        return "("+identificador+","+result+")";
    }


}
```

```java
public class Sumatorio extends Tarea {

    public Sumatorio(int[] valores, String identificador) {
        super(valores, identificador);
    }

    public void operacion() {
        result = 0;
        for (int i=0;i<valores.length;i++){
            result = result + valores[i];
        }
    }

}
```

```java
public class Producto extends Tarea {

    public Producto(int[] valores, String identificador) {
        super(valores, identificador);

    }

    public void operacion() {
        result = 1;
        for (int i=0;i<valores.length;i++){
            result = result * valores[i];
        }
    }

}
```

```java
import java.util.*;

public class Principal {

    public static void main(String[] args) {
        Grid migrid = new Grid();
        Procesador proc1 = new Procesador();
        Procesador proc2 = new Procesador();

        int lista[] = {1,2,3,4,5};

        Sumatorio suma = new Sumatorio(lista,"suma1");
        Producto prod = new Producto (lista,"prod1");

        proc1.setTarea(suma);
        proc2.setTarea(prod);

        migrid.addProcesador(proc1);
        migrid.addProcesador(proc2);

        migrid.addTarea(suma);
        migrid.addTarea(prod);

        migrid.resolverTareas();

        List resultados = migrid.getTareas();
        Iterator it = resultados.iterator();
        while (it.hasNext()){
            System.out.println(it.next());
        }


    }

}
```

Solución básica 2: Interfaces

```java
public class Tarea {
    protected int valores[];
    protected int result;
    private String identificador;
    private Operacion op;

    public Tarea(int[] valores, String identificador, Operacion op) {
        this.valores = valores;
        this.identificador = identificador;
        this.op = op;
    }

    public void operacion(){
        result = op.operar(valores);
    }

    public int getResult() {
        return result;
    }

    public String toString(){
        return "("+identificador+","+result+")";
    }
```

```java
public interface Operacion {
    public int operar(int valores[]);

}

public class Sumatorio implements Operacion {


    public int operar(int[] valores) {
        int result = 0;
        for (int i=0;i<valores.length;i++){
            result = result + valores[i];
        }
        return result;
    }
}

public class Producto implements Operacion {



    public int operar(int[] valores) {
        int result = 0;
        for (int i=0;i<valores.length;i++){
            result = result * valores[i];
        }
        return result;
    }

}
```

```java
import java.util.*;

public class Principal {

    public static void main(String[] args) {
        Grid migrid = new Grid();
        Procesador proc1 = new Procesador();
        Procesador proc2 = new Procesador();

        int lista[] = {1,2,3,4,5};

        Tarea suma = new Tarea(lista,"suma1",new Sumatorio());
        Tarea prod = new Tarea (lista,"prod1", new Producto());

        proc1.setTarea(suma);
        proc2.setTarea(prod);

        migrid.addProcesador(proc1);
        migrid.addProcesador(proc2);

        migrid.addTarea(suma);
        migrid.addTarea(prod);

        migrid.resolverTareas();

        List resultados = migrid.getTareas();
        Iterator it = resultados.iterator();
        while (it.hasNext()){
            System.out.println(it.next());
        }
```

Preguntas

- ¿ Donde hay polimorfismo o sustitución de tipos ?
  - Procesador.setTarea(Tarea t) acepta cualquier subtipo de tarea (solución 1)
  - En el constructor de Tarea aceptamos cualquier implementación de Operación (Solucion 2)
- ¿ Donde hay ligadura dinámica ?
  - Procesador.resolverTarea (Solución 1)
  - Tarea.operar (Solución 2)
- ¿ Es necesario un '**if**' para ejecutar diferentes operaciones ?

Mejoras para subir nota:

1) Si en la lista que se le pasa a la tarea hay un tipo no compatible, al intentar resolver la tarea no podrá. Así que si mas tarde se quiere acceder a la tarea para ver resultado debería salir una excepción. **EXCEPCIONES**

2) Se debería poder listar los procesadores bien ordenados por capacidad de proceso, o por memoria RAM (**COMPARATOR**)

3) Se debería poder obtener de manera eficiente del Grid un procesador utilizando su identificador. **HASHMAP**

4) Permitir que la tarea pueda operar sobre diferentes tipos de datos (enteros, string). Por ejemplos podria tener una tarea con lista ("abc,'def','pepepep') y la operación maximalongitud que devuelva el elemento de mayor longitud de la lista. **OBJECT [ver ejemplo del Map en Java]**

Sistema de directorios

Se trata de modelar en un lenguaje orientado a objetos (Java) el problema de gestión de sistema de directorios. Un directorio tiene un nombre y una lista de ficheros y directorios debajo de el. Un fichero tiene un nombre, extension, tamaño y fecha de modificación. Sobre el directorio podremos añadir ficheros y directorios, eliminar ficheros o directorios, listar ficheros o directorios y buscar ficheros o directorios. Además el administrador del sistema de directorios quiere que se puedan aplicar **filtros** sobre la información. Por ejemplo, un filtro podría retornar todos los ficheros con extension txt, otro filtro podria retornar todos los ficheros de tamaño mayor a 100. Otro filtro podría ser un antivirus que busque ficheros con nombres especiales (virus, soymalo, yaveras). **El sistema debe permitir aplicar varios filtros a la vez.** Es decir, seria absurdo hacer un recorrido por cada filtro, sino que se debe hacer un único recorrido ejecutando todos los filtros. Cada filtro tendrá una lista de resultados una vez ejecutado por el Directorio (aplicarFiltros, añadirFiltro, eliminarFiltro). Crear un programa con un main que pruebe la funcionalidad del Directorio **(crear directorio, crear ficheros, añadir ficheros a directorio, crear filtros, añadir filtros, aplicarFiltros, verResultados).**

Relaciones entre clases

- Clases: Directorio, Fichero, Filtro?
- Relaciones entre clases:
  - Un Directorio tiene 1..N ficheros
  - Un Directorio tiene 1..N Filtros
  - Queremos crear diferentes tipos de filtro !!!! –> Herencia abstracta o de interfaces
    - Ver ejemplo del Comparator o solucionar el ejemplo de la función Filter en Java

Mejoras para subir nota:

1) Si un filtro no tiene ningun resultado el método obtener lista resultados debe devolver una excepción. **EXCEPCION**

2) Se debería permitir listar los ficheros bien ordenados por nombre o tamaño (**COMPARATOR**).

3) Se debería permitir la recursividad en las búsquedas y aplicación de filtros. Pista, se puede tratar de manera análoga a ficheros y directorios (Composite Pattern)

Preguntas

- ¿ Donde hay polimorfismo o sustitución de tipos ?
  - Directorio.addFiltro(Filtro t) acepta cualquier subtipo de Filtro
- ¿ Donde hay ligadura dinámica ?
  - Mirar Directorio.aplicarFiltros (…)
- ¿ Es necesario un ‘**if**’ para aplicar nuevos filtros ?

Agencia de actores/actrices y figurantes

Se trata de modelar en un lenguaje orientado a objetos (Java) el problema de la selección de personal inscrito en una agencia, para los castings de películas y series de televisión. Estas personas son registradas en la agencia bajo dos roles: a) actores/actrices y b) figurantes. Todos ellos detallan una serie de características en su registro en la agencia. Para simplificar deben aparecer su nombre, edad, sexo, nacionalidad, teléfono de contacto y aspecto.

Cuando a la agencia le llega una petición de personal por parte de una productora, para realizar películas o series de televisión, se les indica el número de personas que se necesitan y los detalles realmente importantes para cada uno de ellos. Por ejemplo, pueden solicitar tres personas de entre 25 y 35 años (edad) de nacionalidad brasileña (nacionalidad), o bien que sea una chica (sexo) con cabello largo (aspecto) y entre 18 y 22 años (edad). Por lo que tan sólo sería necesario filtrar el personal inscrito en la agencia y seleccionar la cantidad necesaria de cada perfil. Se debe crear un programa con un main que automatice la selección de personal de la agencia **(crear agencia, crear actores/actrices y figurantes, crear perfiles, filtrar según perfiles, ver resultados).**

Relaciones entre clases

- Clases: Agencia, Persona, Perfil?
- Relaciones entre clases:
  - Una Agencia tiene 1..N Personas
  - Una Agencia tiene 1..N Perfiles
  - Queremos crear diferentes tipos de filtro !!!! –> Herencia abstracta o de interfaces
    - Ver ejemplo del Comparator o solucionar el ejemplo de la función Filter en Java

Mejoras para subir nota:

1) Si un filtro no tiene ningún resultado, el método obtener lista resultados debe lanzar una excepción. **EXCEPCION**

2) Se debería permitir listar el personal inscrito en la agencia por nombre, edad y nacionalidad. (**COMPARATOR**).

3) Se debería permitir obtener de forma eficiente a cualquier inscrito por su nombre. 4) Se debería permitir más de un detalle en el campo "aspecto" dentro de cada perfil y su aplicación en la operación de filtrado.

Preguntas

- ¿ Donde hay polimorfismo o sustitución de tipos ?
  - Agencia.addPerfil(Perfil t) acepta cualquier subtipo de Perfil
- ¿ Donde hay ligadura dinámica ?
  - Mirar Agencia.filtrar según perfiles (…)
- ¿ Es necesario un '**if**' para filtrar por nuevos Perfiles ?

Conclusiones

- ¿ Sabéis encontrar la herencia, el polimorfismo y la ligadura dinámica en un enunciado ?
- De las tres pruebas de nivel, ¿ creéis que el Map, el Filter o Comparator tienen algún parecido ?