Orientación a Objetos I

2025

Explicación de práctica #5
correspondiente a los ejercicios de la
semana del 29 de Septiembre

Actividades de la semana anterior

- Ejercicio 10: Method lookup con Empleados
- Ejercicio 11: Cuenta con ganchos

Cuadernillo Semestral de Actividades

Actualizado: 25 de septiembre de 2025

Esta semana:

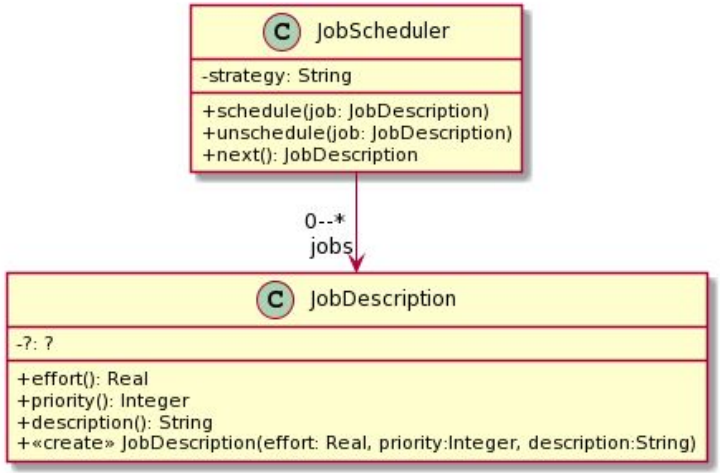
- Ejercicio 12: Job scheduler (retomamos)
- Ejercicio 13: ¡A implementar Inversores!
- Ejercicio 14: Volumen y superficie de sólidos

Ejercicio 12: Job Scheduler

Ejercicios de la semana:

- **Ejercicio 12: Job scheduler**
- Ejercicio 13: A implementar Inversores
- Ejercicio 14: Volumen y superficie de sólidos

El JobScheduler es un objeto cuya responsabilidad es determinar qué trabajo debe resolverse a continuación. El siguiente diseño ayuda a entender cómo funciona la implementación actual del JobScheduler.



- El mensaje `schedule(job: JobDescription)` recibe un job (trabajo) y lo agrega al final de la colección de trabajos pendientes.
- El mensaje `next()` determina cuál es el siguiente trabajo de la colección que debe ser atendido, lo retorna, y lo quita de la colección.

FACULT

```java
public class JobScheduler {
    public JobDescription next() {
        JobDescription nextJob = null;
        switch (strategy) {
            case "FIFO":
                nextJob = jobs.get(0);
                this.unschedule(nextJob);
                return nextJob;
            case "LIFO":
                nextJob = jobs.get(jobs.size()-1);
                this.unschedule(nextJob);
                return nextJob;
            case "HighestPriority":
                nextJob = jobs.stream()
                    .max((j1,j2) -> Double.compare(j1.getPriority(), j2.getPriority()))
                    .orElse(null);
                this.unschedule(nextJob);
                return nextJob;
            case "MostEffort":
                nextJob = jobs.stream()
                    .max((j1,j2) -> Double.compare(j1.getEffort(), j2.getEffort()))
                    .orElse(null);
                this.unschedule(nextJob);
                return nextJob;
        }
        return null;
    }
}
```

```java
public class JobScheduler {
    public JobDescription next() {
        JobDescription nextJob = null;
        switch (strategy) {
            case "FIFO":
                nextJob = jobs.get(0);
                this.unschedule(nextJob);
                return nextJob;
            case "LIFO":
                nextJob = jobs.get(jobs.size()-1);
                this.unschedule(nextJob);
                return nextJob;
            case "HighestPriority":
                nextJob = jobs.stream()
                    .max((j1,j2) -> Double.compare(j1.getPriority(), j2.getPriority()))
                    .orElse(null);
                this.unschedule(nextJob);
                return nextJob;
            case "MostEffort":
                nextJob = jobs.stream()
                    .max((j1,j2) -> Double.compare(j1.getEffort(), j2.getEffort()))
                    .orElse(null);
                this.unschedule(nextJob);
                return nextJob;
        }
        return null;
    }
}
```

- Secuencia de ifs (o sentencia switch/case) para implementar alternativas de un mismo comportamiento.

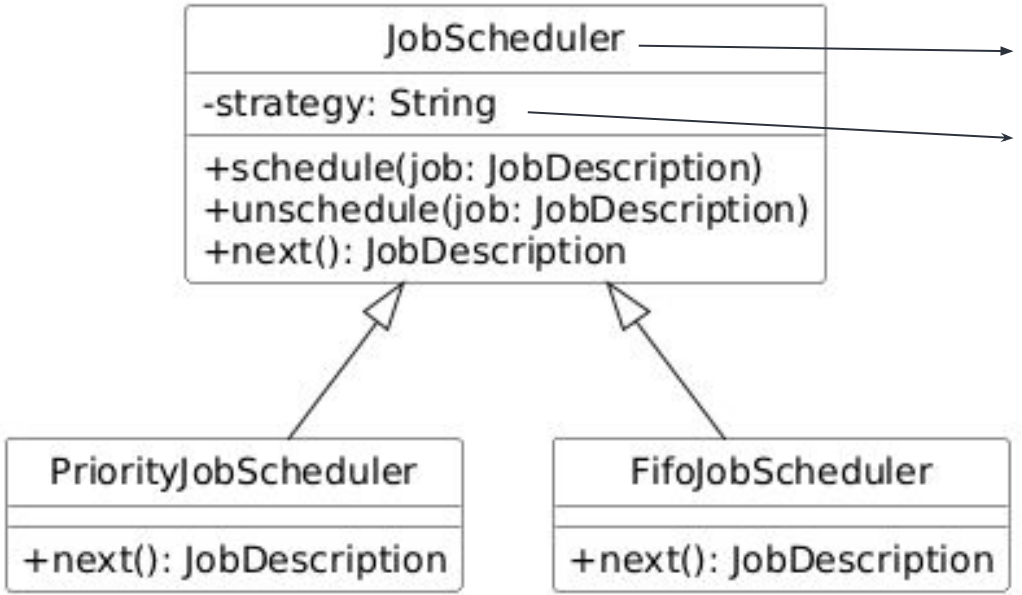
- ¿Código duplicado?
Para pensar y lo retomamos la semana que viene

```java
public class JobScheduler {
    public JobDescription next() {
        JobDescription nextJob = null;
        switch (strategy) {
            case "FIFO":
                nextJob = jobs.get(0);
                this.unschedule(nextJob);
                return nextJob;
            case "LIFO":
                nextJob = jobs.get(jobs.size()-1);
                this.unschedule(nextJob);
                return nextJob;
            case "HighestPriority":
                nextJob = jobs.stream()
                    .max((j1,j2) -> Double.compare(j1.getPriority(), j2.getPriority()))
                    .orElse(null);
                this.unschedule(nextJob);
                return nextJob;
            case "MostEffort":
                nextJob = jobs.stream()
                    .max((j1,j2) -> Double.compare(j1.getEffort(), j2.getEffort()))
                    .orElse(null);
                this.unschedule(nextJob);
                return nextJob;
        }
        return null;
    }
}
```

¿Cómo usamos lo que aprendimos de herencia y polimorfismo para resolver estos problemas?

```java
public class JobScheduler {
    public JobDescription next() {
        JobDescription nextJob = null;
        switch (strategy) {
            case "FIFO":
                nextJob = jobs.get(0);
                this.unschedule(nextJob);
                return nextJob;
            case "LIFO":
                nextJob = jobs.get(jobs.size()-1);
                this.unschedule(nextJob);
                return nextJob;
            case "HighestPriority":
                nextJob = jobs.stream()
                    .max((j1,j2) -> Double.compare(j1.getPriority(), j2.getPriority()))
                    .orElse(null);
                this.unschedule(nextJob);
                return nextJob;
            case "MostEffort":
                nextJob = jobs.stream()
                    .max((j1,j2) -> Double.compare(j1.getEffort(), j2.getEffort()))
                    .orElse(null);
                this.unschedule(nextJob);
                return nextJob;
        }
        return null;
    }
}
```

Diagrama INCOMPLETO - Revisar y completar!



¿Clase abstracta o concreta?

¿Hace falta esta variable?

Este nuevo diseño, ¿afecta a los test?

. . .

FACULTAD DE INFORMATICA

UNIVERSIDAD NACIONAL DE LA PLATA

Comenzando con la clase FifoJobScheduler …

```
1 public class FifoJobScheduler extends JobScheduler {
2     @Override
3     public JobDescription next() {
4         JobDescription nextJob = this.jobs.get(0);
5         this.unschedule(nextJob);
6         return nextJob;
7     }
8 }
9
```

¿Que pasa con el test?

```
1 public class JobSchedulerTest {
2     private JobScheduler newFifoScheduler() {
3         JobScheduler fifoScheduler = new JobScheduler();
4         fifoScheduler.setStrategy("FIFO");
5         return fifoScheduler;
6     }
7 }
```

Instanciamos la subclase FifoJobScheduler, pero…

```
1 private JobScheduler newFifoScheduler() {
2     JobScheduler fifoScheduler = new FifoJobScheduler();
3     fifoScheduler.setStrategy("FIFO");
4     return fifoScheduler;
5 }
6
```

¿Es necesario “setearle la estrategia”?

FifoJobScheduler ya *es-un* Scheduler que trabaja con estrategia “FIFO”

```
1 private JobScheduler newFifoScheduler() {
2     JobScheduler fifoScheduler = new FifoJobScheduler();
3     fifoScheduler.setStrategy("FIFO");
4     return fifoScheduler;
5 }
6
```

```java
private JobScheduler newFifoScheduler() {
    JobScheduler fifoScheduler = new FifoJobScheduler();
    return fifoScheduler;
}
```

Bonus: nos podemos ahorrar la variable si queremos

```java
private JobScheduler newFifoScheduler() {
    return new FifoJobScheduler();
}
```

Ejercicio 13: ¡A implementar Inversores!

Ejercicios de la semana:

- Ejercicio 12: Job scheduler
- **Ejercicio 13: ¡A implementar Inversores!**
- Ejercicio 14: Volumen y superficie de sólidos

Lo vemos en la teoría la semana próxima!

FACU

Ejercicio 13: ¡A implementar Inversores!

Retomando el Ejercicio 5, trabajamos en el diseño y modelado UML de un sistema de inversiones, donde definimos las clases, atributos y asociaciones necesarias para representar inversores y sus diferentes tipos de inversiones. Ahora es el momento de llevar el diseño a la práctica: vamos a implementar en Java lo diseñado y asegurar su calidad mediante pruebas automatizadas.

Tareas:

1. **Implemente**
   a. Realice el mapeo del modelo conceptual, a un diagrama de clases de UML.
   b. Implemente en Java lo necesario para que se pueda conocer el valor actual de cada inversión. Y también el monto total de las inversiones realizadas por un inversor.
2. **Pruebas automatizadas**
   a. Diseñe los casos de prueba teniendo en cuenta los conceptos de valores de borde y particiones equivalentes vistos en la teoría.
   b. Implemente utilizando JUnit los tests automatizados diseñados en el punto anterior.
3. **Discuta con el ayudante**
   a. Consulte con un ayudante los casos de prueba diseñados

Ejemplo de un test en Java

```java
public class PersonaTest {
    private Persona james, guido;

    @BeforeEach
    void setUp() throws Exception {
        james = new Persona();
        guido = new Persona();
        james.setApellido("Glosing");
        james.setNombre("James");
        guido.setApellido("van Rossum");
        guido.setNombre("Guido");

    }

    @Test
    public void testNombreCompleto() {
        assertEquals("Glosing, James", james.getNombreCompleto());
        assertEquals("van Rossum, Guido", guido.getNombreCompleto());
    }
}
```

Necesario para que lo reconozca el ambiente

Creamos los objetos que vamos a probar

Verificamos los resultados esperados con “assertEquals”

Ejercicio 14: Volumen y superficie de sólidos

Ejercicios de la semana:

- Ejercicio 12: Job scheduler
- Ejercicio 13: ¡A implementar Inversores!
- **Ejercicio 14: Volumen y superficie de sólidos**

Una empresa siderúrgica quiere introducir en su sistema de gestión nuevos **cálculos de volumen y superficie** exterior para las piezas que produce.

El **volumen** le sirve para determinar **cuánto material** ha utilizado.

La **superficie exterior** le sirve para determinar la **cantidad de pintura** que utilizó para pintar las piezas.

FACULTAD DE INFORMATICA

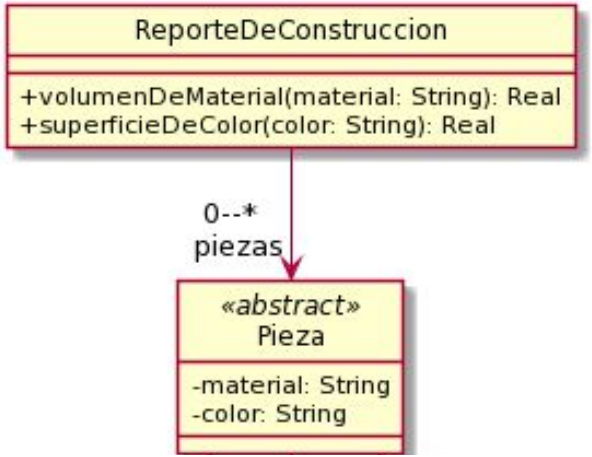
UNIVERSIDAD NACIONAL DE LA PLATA

Ejercicio 14: Volumen y superficie de sólidos

El siguiente diagrama UML muestra el diseño actual del sistema.

En el mismo puede observarse que un **ReporteDeConstruccion** tiene la lista de las piezas que fueron construidas.

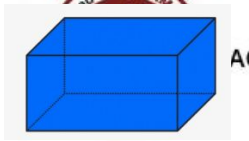
Pieza es una clase abstracta.
¿Qué subclases debería tener?

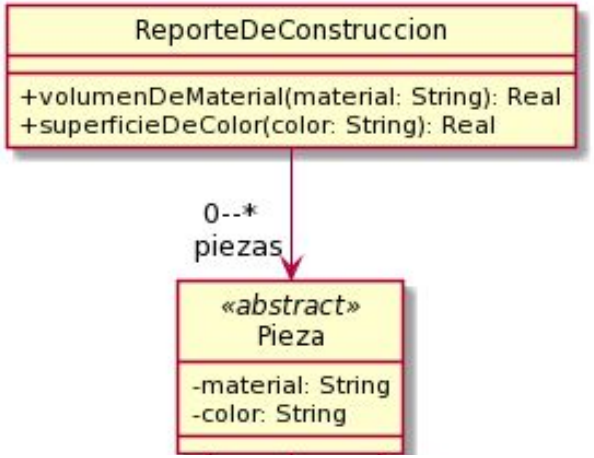


Ejercicio 14: Volumen y superficie de sólidos

Esperaríamos usar el reporte así:

reporte.volumenDeMaterial('Hierro');
reporte.superficieDeColor('Rojo');





Ejercicio 14: Volumen y superficie de sólidos

Complete el diseño e implemente

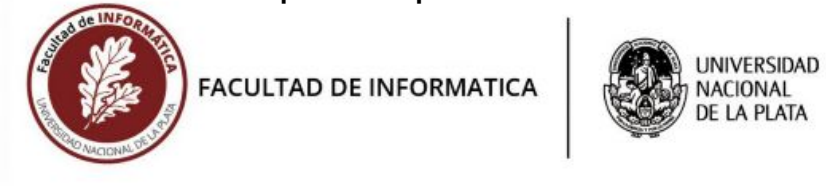
Su tarea es completar el diseño e implementarlo siguiendo las especificaciones que se exponen a continuación:

volumenDeMaterial(nombreDeMaterial: String)

> "Recibe como parámetro un nombre de material (un string, por ejemplo 'Hierro'). Retorna la suma de los volúmenes de todas las piezas hechas en ese material"

superficieDeColor(unNombreDeColor: String)

> "Recibe como parámetro un color (un string, por ejemplo 'Rojo'). Retorna la suma de las superficies externas de todas las piezas pintadas con ese color".



Ejercicio 14: Volumen y superficie de sólidos

Discuta con el ayudante

Es probable que note una similitud entre este ejercicio y el de "Figuras y cuerpos" que realizó anteriormente, ya que en ambos se pueden construir cilindros y prismas rectangulares. Sin embargo las implementaciones varían.

Enumere las diferencias y similitudes entre ambos ejercicios y luego consulte con el ayudante.

Foros de consulta

Cómo preguntar en el foro

Breve guía para poder sacar el mejor provecho al foro y a la convivencia a través de las preguntas y respuestas.

FACULTAD DE INFORMATICA | UNIVERSIDAD NACIONAL DE LA PLATA