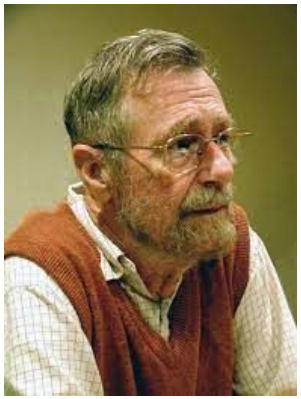
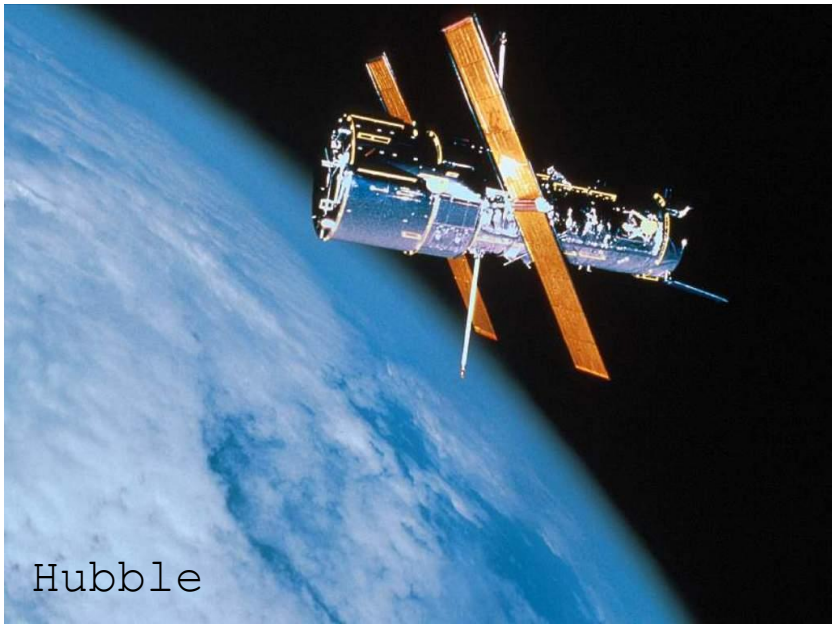


“If debugging is the process of removing software bugs, then programming must be the process of putting them in.”
— **Edsger W. Dijkstra**

“If (debugging == fork(remove software bugs){
programming = fork(add software bugs)
}.”
— **Edsger W. Dijkstra**



Ariane 5 con componentes de Ariane 4

Contexto

- En SE carencia de errores es una métrica de calidad
- Testing es una manera de buscar mejorar la calidad:
  - Test de Stress
  - Tests de aceptación
  - Tests de integración
  - Test de Regresión
  - T**ests de unidad**

Test de unidad

Testeo de la *mínima unidad de ejecución*.

En OOP, la mínima unidad es un método.

Objetivo: aislar cada parte de un programa y mostrar que funciona correctamente.

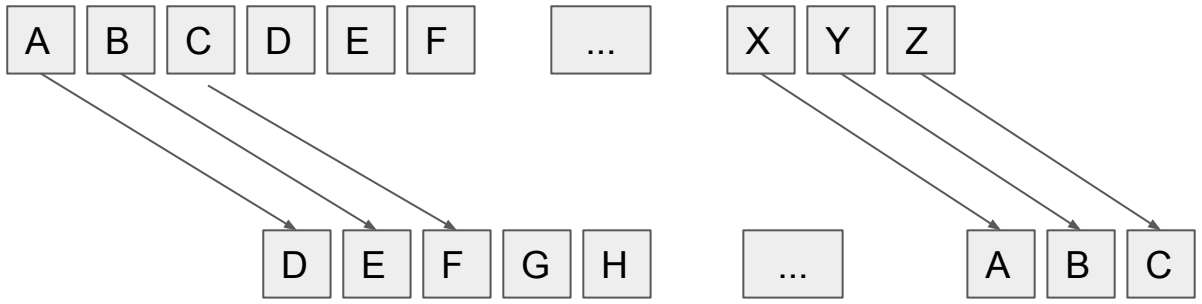
Cada test confirma que un método produce el output esperado ante un input conocido.

Es como un contrato escrito de lo que esa unidad tiene que satisfacer.

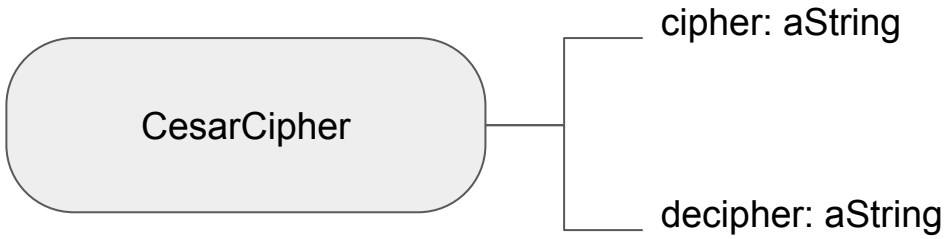
Cifrado de Texto

Un texto cifrado es el resultado de aplicar un algoritmo que ha ofuscado un texto original (plain text)

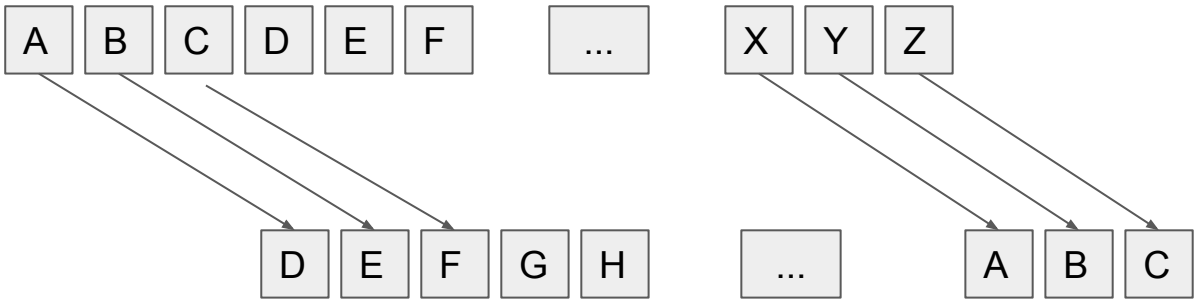
El cifrado de César es un cifrado de sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto.



10 minutos…



Defina 5 casos de prueba que usaría para verificar si una implementación de CesarCipher (con salto 3) es correcta



Paradoja del Testing

- Escribir casos de testing es deseable
- Escribir casos de testing costoso y aburrido
- Testear todos los métodos no es práctico

```
public String getName(){
  return firstName + lastName;
}
```

Objetivo: min. los casos y max. 'covertura'

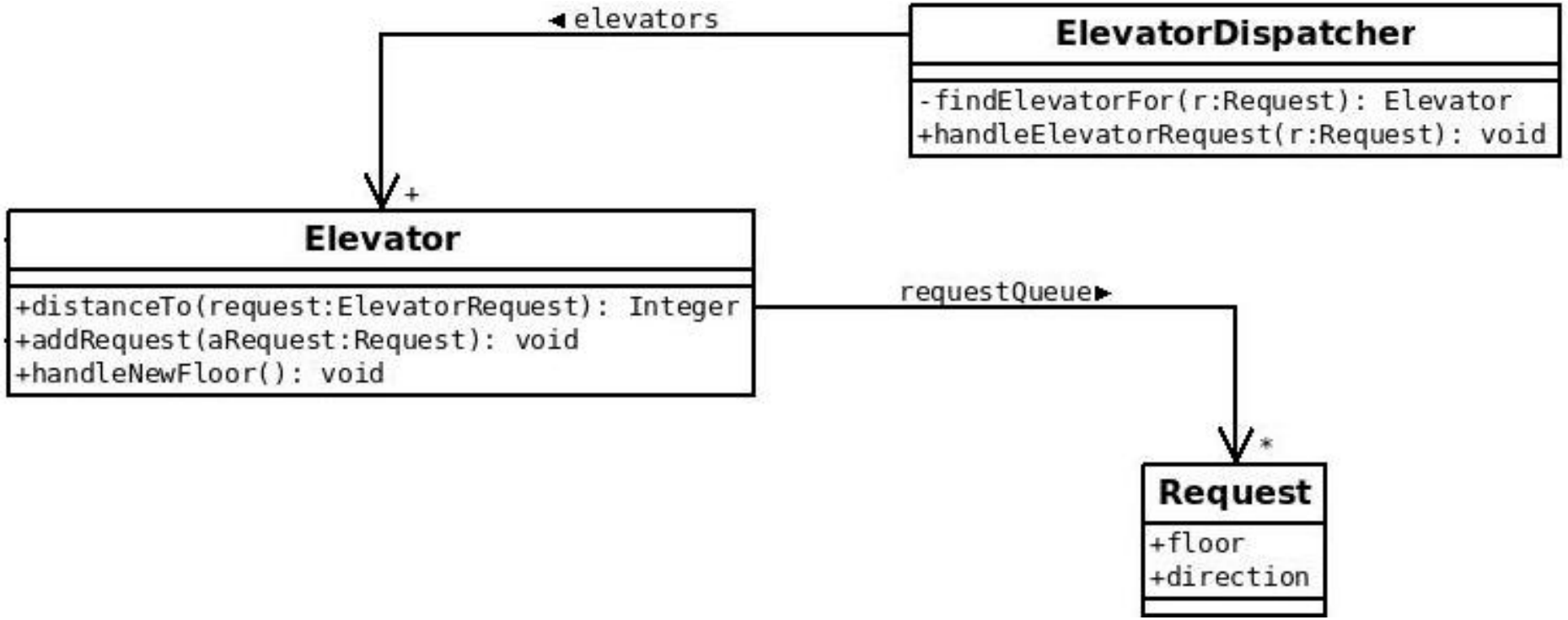
Estratégias para Diseñar Casos de Test

- Particiones Equivalentes
  - Considerar el “dominio” de la funcionalidad a testear
    - Si un elemento pasa el test, otros del mismo conjunto pasarán
  - Considerando el complemento del dominio
    - No hace lo que no debe hacer
- Valores de Borde
  - Considerando las Particiones Equivalentes
    - Se testean escenarios con valores en los límites de las particiones

```java
package ar.edu.unlp.info.oo2.tp;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import ar.edu.unlp.info.oo2.tp.roo2.InterfaceFramework;
import ar.edu.unlp.info.oo2.tp.roo2.VigenereCipher;

public class CesarTest {
    VigenereCipher cesar1;
    VigenereCipher cesar2;


    @BeforeEach
    void setUp() {
        InterfaceFramework f = new InterfaceFramework();
        this.cesar1 = f.crearCesar(3);
        this.cesar2 = f.crearCesar("123456789", 3);

    }

    @Test
    void testCipherEquals() {
        assertEquals(cesar1.cipher("hola"), "krod");
        assertEquals(cesar1.cipher("xyz"), "abc");
        assertEquals(cesar2.cipher("1234"), "4567");
        assertEquals(cesar2.cipher("789"), "123");

    }
```

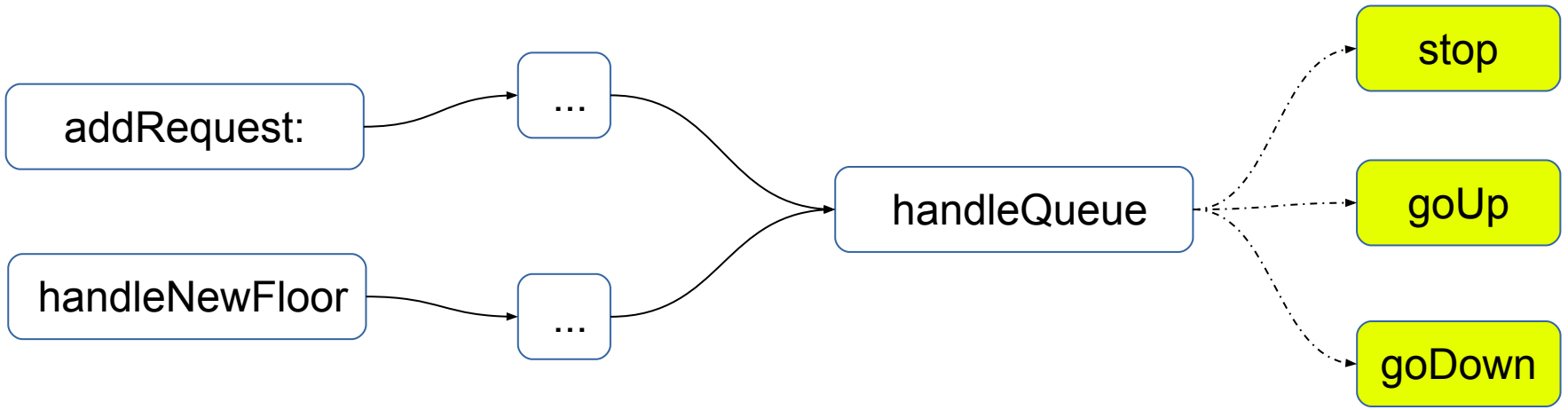
```java
@Test
public void testCipherConNull(){
    assertThrows(NullPointerException.class,()->{
        cesar1.cipher(null);} );
}


@Test
public void testDecipherConNull(){
    assertThrows(NullPointerException.class, ()->{
        cesar1.decipher(null);} );
}
```

Ascensores



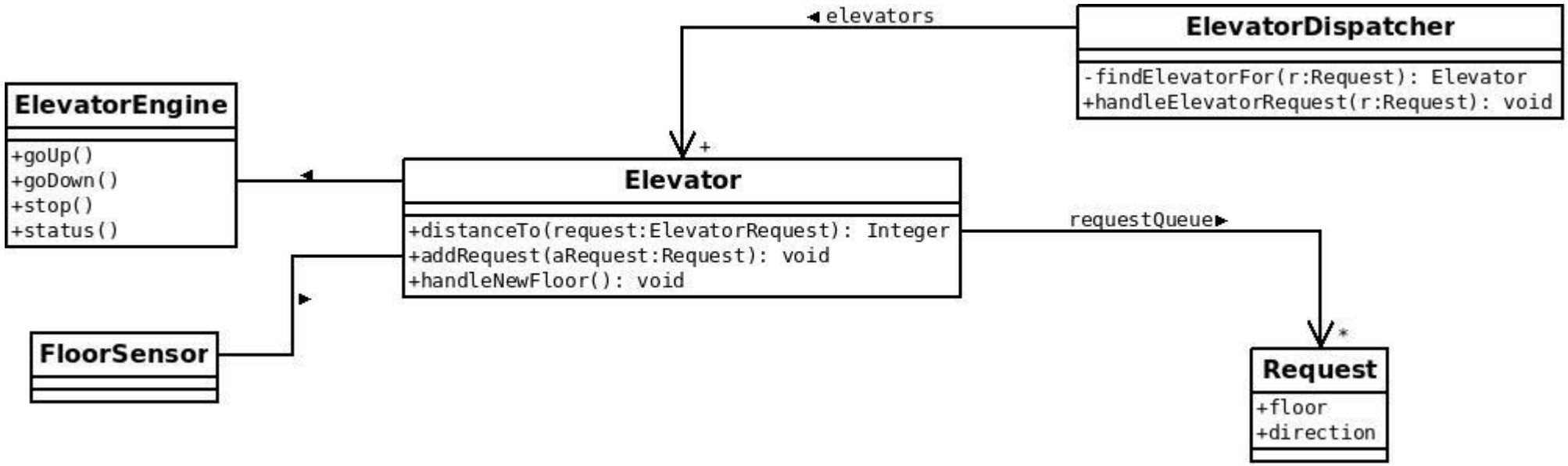
Cómo hace el ascensor para moverse?
Cómo hace para el ascensor saber en que piso está?

Ascensor




```
handleQueue
    requestQueue first floor = currentFloor
        ifTrue: [
            self
                stop;
                firstRequestDone ]
        ifFalse: [
            requestQueue first floor < currentFloor
                ifTrue: [ self goDown ]
                ifFalse: [ self goUp ] ]
```

Ascensores



ElevatorEngine y FloorSensor dependen del sistema real (SUT)

Test Double

Xunit Test Patterns

-G. Meszaros-

Test Double

- Problema general
  - Es necesario realizar pruebas de un “SUT” que depende de un módulo u objeto
  - El módulo u objeto requerido no se puede utilizar en el ambiente de la pruebas
  - Las pruebas pueden ejercitar
    - Configuraciones válidas del sistema
    - “Salidas indirectas” del sistema
    - Lógica del sistema
    - Protocolos
- Test Double es un lenguaje de patrones

Lenguajes de Patrones

- Test Double
  - Crear un objeto que es una maqueta (polimórfica) del objeto o módulo requerido
  - Utilizar la maqueta según se necesite
- Rangos de implementación
  - Cascarón vacío → Simulación
- Se generan diferentes patrones que se aplican a cada caso

Test Double

- Patrones
  - **Test Stub**: cascarón vacío. Sirve para que el SUT envíe los mensajes esperados
  - **Test Spy**: Test Stub + registro de outputs
  - **Mock Object**: test Stub + verification of outputs
  - **Fake Object**: imitación. Se comporta como el módulo real (protocolos, tiempos de respuesta, etc)

Ascensores

- Métodos goUp, goDown, stop
  - Invocan mensajes del “motor”
  - Recibe eventos del “sensor de piso”

Con cada TestDouble se puede:

- **Stub**: recibe los mensajes que envía el Ascensor
  - testear distanceTo(:Request)
- **Spy**: guarda registro de los comando del Ascensor
  - verificar los métodos invocados en el motor
- **Mock**: comprueba la validez de los comandos que envía el Ascensor
- **Fake**: simula el comportamiento del motor → generando eventos del sensor de piso

Test Double

- Implementar clases según sea necesario
  - Objetos que no están disponibles para probar
- Que ocurre con objetos que están disponibles
  - Respaldar con test cases
- Implementación:
  - Test Stub: simple y barato
  - Fake object: demanda análisis, threading, requiere mantenimiento

Comm Check...: The Final Flight of Shuttle Columbia

Michael Cabbage, William Harwood, 2008