Calidad, Validación y Verificación del Software

Técnicas Basadas en la Especificación: Clases de Equivalencia (Parte 2)
Diseño e Implementación

Grado en Ingeniería Informática del Software

Javier Tuya, Raquel Blanco
Grupo de Investigación en Ingeniería del Software
http://giis.uniovi.es

Curso 2021-2022

Contenidos

- Parte 1 (Conceptos básicos):
  - Técnica Básica: Partición en Clases de Equivalencia
  - Técnica Complementaria: Análisis de Valores Límite
  - Estrategia de combinación de clases – Jerarquía
- Parte 2 (Diseño, implementación y automatización)
  - Unitarias: Sin interfaz de usuario ni base de datos
  - Unitarias: Con base de datos
  - Integración con el interfaz de usuario
  - (+Automatización Java/Swing y Spring Boot)
  - Resumen
- Parte 3 (Otras):
  - Otras técnicas. Explosión combinatoria
    - Tablas de Decisión
    - Árbol de Clasificación
    - Técnicas Combinatorias
  - Validaciones de Datos

Contenidos vs. tecnologías

Ejemplo	samples-test-java		samples-test-spring	
Tecnología	Java/Swing		Spring Boot	
Herramienta	Junit	JBehave (BDD)	Junit	JBehave (BDD)
Qué se prueba:				
Función independiente	*.ut.*Function*	*.ut.jbehave.*		
Función con BD	*.ut.*Database	*.ut.jbehave.*		
Función con BD y parámetros	*.ut.*Database	*.ut.jbehave.*	*.ut.*Repository *.ut.*Parametrized	*.ut.jbehave.*
			*.ut.*Mock *.ut.*WebController	
			*.ut.*RestService	
Interfaz Usuario	*.it.* (+AssertJ)	*.it.jbehave.*	*.it.* (+Selenium)	*.it.jbehave.*

Proyectos Ejemplo:

- https://in2test.lsi.uniovi.es/tools/samples-test-java/
- https://in2test.lsi.uniovi.es/tools/samples-test-spring/

Función independiente
Ejercicio

- Problema 3a: Se tiene una función que determina el descuento a aplicar en las compras mediante tarjeta. Los descuentos se determinan como se indica a continuación y son acumulables: Si el cliente acaba de abrir una cuenta de crédito obtiene el 15% de descuento en todas sus compras de hoy, si es un cliente habitual con tarjeta de fidelización obtiene un 10% de descuento. Si el cliente tiene un cupón de descuento obtiene el 20% de descuento (no acumulable con el descuento de nuevo cliente).

Función independiente

Diseño

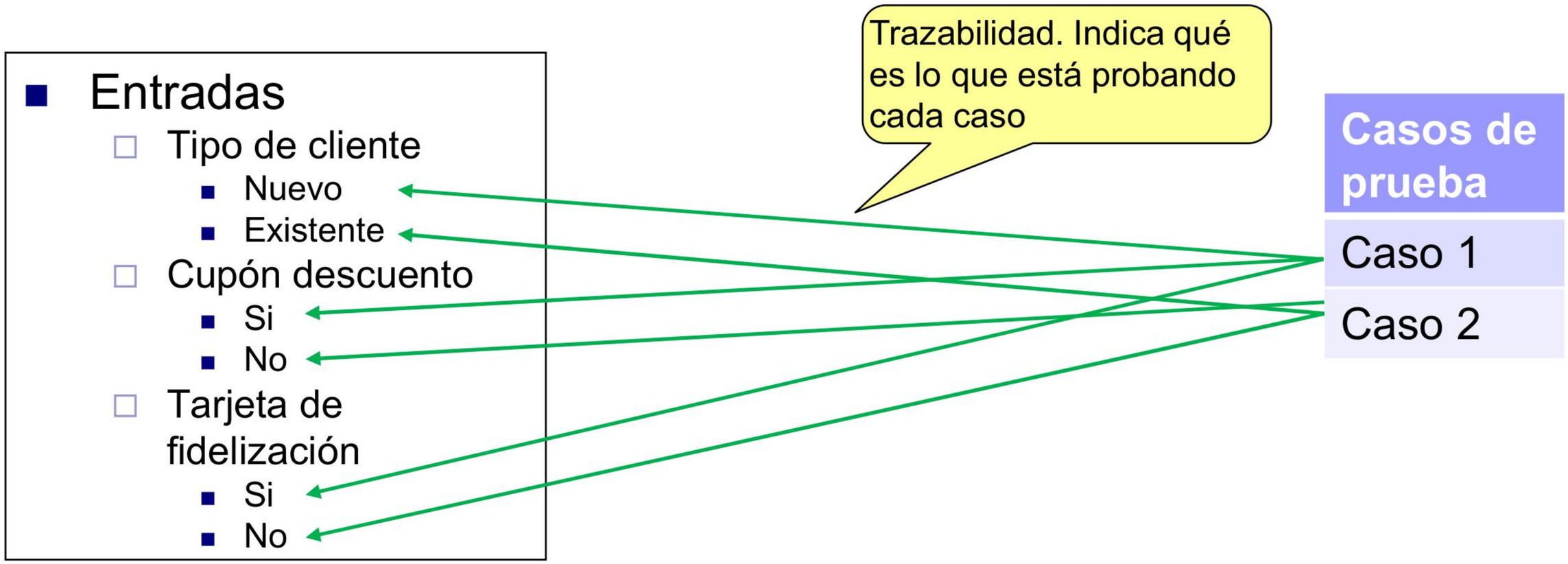
- Solución (1):
  - Entradas
    - Tipo de cliente
      - Nuevo
      - Existente
    - Cupón descuento
      - Si
      - No
    - Tarjeta de fidelización
      - Si
      - No
  - Salidas
    - Acumulación de descuentos
      - Se acumulan
      - No se acumulan
    - Descuento aplicado (%)
      - 15; 10; 20; 0

Generamos suficientes casos de prueba usando el “minimized approach”?
Combinamos de froma exhaustiva para obtener todas las posibilidades?

Función independiente

Diseño – Each choice

- Antes de derivar los casos de prueba debemos decidir la estrategia de combinación.
  - Parte ESENCIAL del diseño de las pruebas: permitirá realizar pruebas más o menos exhaustivas
- Each Choice (Estrategia minimizada): Estrategia por defecto
  - Implica no combinar: Se han de cubrir todas las situaciones con el menor número de casos de prueba (deja libertad a la hora de establecer los casos)



Función independiente

Diseño - Multiple Combination

- Combinar todo con todo (fuerza bruta)
  - Se han de cubrir todas las combinaciones de todas las situaciones
  - Transformar las situaciones a cubrir para reflejar estas combinaciones

- Entradas
  - Tipo de cliente
    - Nuevo
    - Existente
  - Cupón descuento
    - Si
    - No
  - Tarjeta de fidelización
    - Si
    - No

- Entradas (combinación múltiple)
  - Nuevo, sin cupón, sin tarjeta
  - Nuevo, con cupón, sin tarjeta
  - Nuevo, sin cupón, con tarjeta
  - Nuevo, con cupón, con tarjeta
  - Existente, sin cupón, sin tarjeta
  - Existente, con cupón, sin tarjeta
  - Existente, sin cupón, con tarjeta
  - Existente, con cupón, con tarjeta

Casos de prueba
Caso 1
Caso 2
Caso 3
Caso 4
Caso 5
Caso 6
Caso 7
Caso 8

Función independiente

Diseño - Base Choice

- Permite probar la “sensibilidad” ante pequeños cambios:
  - Elegir una combinación base (p.e. la más típica o importante)
  - El resto son igual a la base salvo un cambio en una de las situaciones

- Entradas
  - Tipo de cliente
    - Nuevo
    - Existente
  - Cupón descuento
    - Si
    - No
  - Tarjeta de fidelización
    - Si
    - No

- Entradas (base choice)
  - Existente, sin cupón, sin tarjeta
  - Nuevo, sin cupón, sin tarjeta
  - Existente, con cupón, sin tarjeta
  - Existente, sin cupón, con tarjeta

Casos de prueba
Caso 1
Caso 2
Caso 3
Caso 4

Función independiente

Diseño - Combinaciones parciales

- Algunas test conditions son más importantes que otras
  - Combinar de formas diferentes
  - Ejemplo, para cada valor de tipo de cliente, combinar usando base choice

- Entradas
  - Tipo de cliente
    - Nuevo
    - Existente
  - Cupón descuento
    - Si
    - No
  - Tarjeta de fidelización
    - Si
    - No

- Entradas (base choice para cada tipo cliente)
  - Nuevo
    - Sin cupon, sin tarjeta
    - Con cupón, sin tarjeta
    - Sin cupón, con tarjeta
  - Existente
    - Sin cupon, sin tarjeta
    - Con cupón, sin tarjeta
    - Sin cupón, con tarjeta

Casos de prueba
Caso 1
Caso 2
Caso 3
Caso 4
Caso 5
Caso 6

Función independiente

Diseño

- Solución 2 (la jerarquía implica combinaciones parciales):
  - Entradas (Combinación parcial tipo cliente y cupones/fidelización)
    - Nuevo Cliente
      - Sin cupón descuento ni tarjeta fideliz.
      - Con cupón descuento
      - Con tarjeta fidelización (inválida)
    - Cliente existente
      - Sin cupón descuento ni tarjeta fideliz.
      - Con cupón descuento
      - Con tarjeta de fidelización
  - Salidas
    - Acumulación de descuentos
      - Se acumulan
      - No se acumulan
    - Descuento aplicado (%)
      - 15
      - 10
      - 20
      - 0

Discusión:

Inicialmente, podemos pensar directamente de otra forma:

Una sola clase de entrada: tipo de cliente, que luego se subdivide

Cuáles serían los casos de prueba?

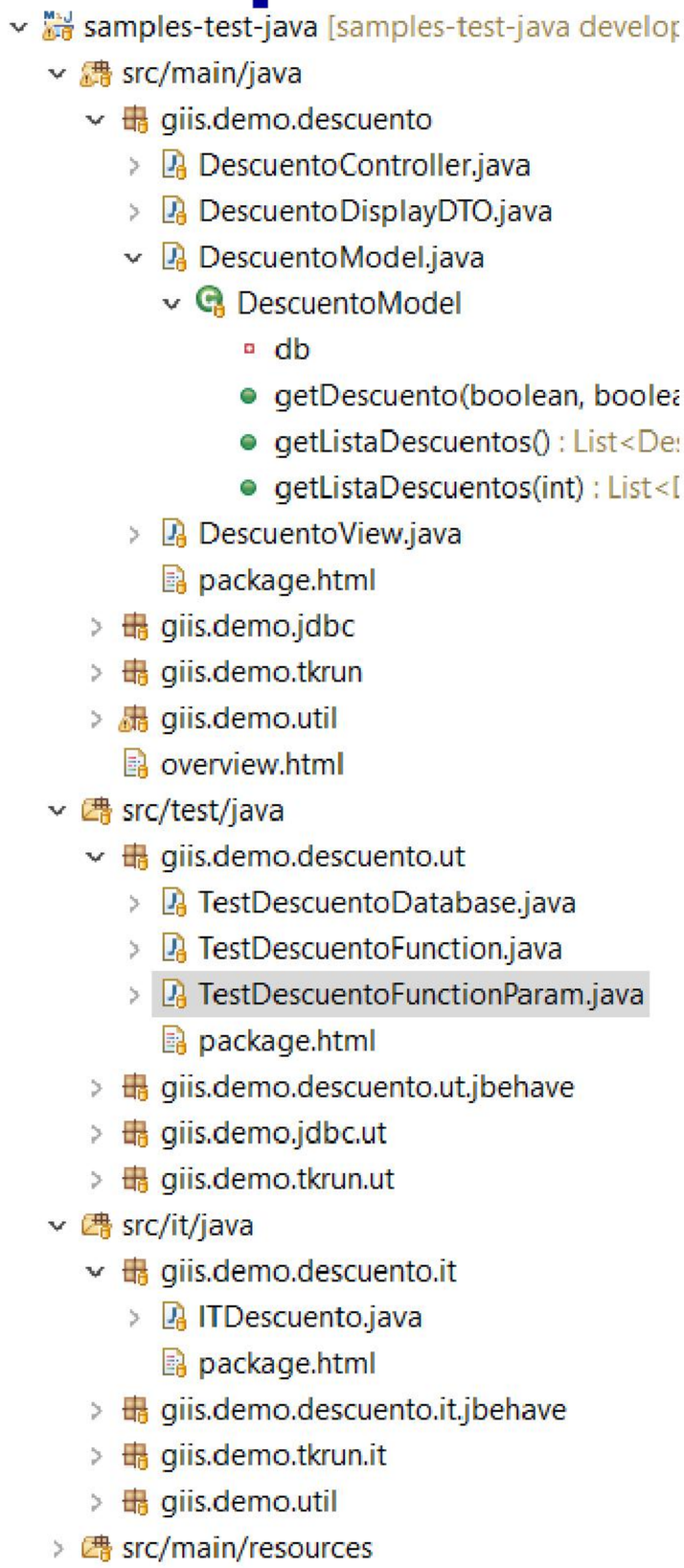
Función independiente

Implementación (casos de prueba+trazabilidad)

- Entradas (Combinaciones tipo cliente y cupones/fidelización)
  - Nuevo Cliente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta fidelización (inválida)
  - Cliente existente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta de fidelización
- Salidas
  - Acumulación de descuentos
    - Se acumulan
    - No se acumulan
  - Descuento aplicado (%)
    - 15
    - 10
    - 20
    - 0

Casos de prueba	Salida Esperada (% Dto.)
Caso 1	15
Caso 2	20
Caso 3	Error?
Caso 4	0
Caso 5	20
Caso 6	10
Caso 7	30

Implementación Sistema a Probar (SUT)



- Estructura proyecto Maven
  - src/main/java
    - Aplicación Java/swing
    - getListaDescuentos()
    - …
  - src/test/java
    - Pruebas unitarias
    - testDescuentoFunction()
    - …
  - src/it/java (requiere config. específica Maven)
    - Pruebas interfaz usuario
- Targets Maven (pom.xml)
  - mvn test: pruebas unitarias
  - mvn verify: +pruebas interfaz usuario
  - mvn install: +otros reports

Función independiente

Implementación (automatización con JUnit)

Definir estrategia para agrupar casos de prueba en diferentes métodos
- un caso por método -> demasiados tests?
- muchos casos en el mismo método -> demasiadas dependencias si falla uno?

```
1  package giic.demo.descuento.ut;
2⊕ impor samples-test-java/index.html
10
12⊕ * Pruebas del ejemplo de descuentos de clientes (Problema 3a).
36 public class TestDescuentoFunction {
37⊖     @Test
38      public void testClientesNuevos() {
39          DescuentoModel model=new DescuentoModel();
40          assertEquals("caso 1",15,model.getDescuento(true, false, false));
41          assertEquals("caso 2",20,model.getDescuento(true, true, false));
42      }
43⊖     @Test
44      public void testClientesHabituales() {
45          DescuentoModel model=new DescuentoModel();
46          assertEquals("caso 4",0,model.getDescuento(false, false, false));
47          assertEquals("caso 5",20,model.getDescuento(false, true, false));
48          assertEquals("caso 6",10,model.getDescuento(false, false, true));
49          assertEquals("caso 7",30,model.getDescuento(false, true, true));
50      }
52⊕      * Prueba de la clase invalida (causa excepcion)
54⊖     @Test(expected=ApplicationException.class)
55      public void testClientesNuevosNoPuedenTenerTarjeta() {
56          DescuentoModel model=new DescuentoModel();
57          model.getDescuento(true, false, true); //caso 4
58      }
```

En teoría, no más de un requisito o característica por método
Aquí se ha agrupado por:
- para nuevos
- para habituales

Excepciones (clases inválidas)
Separadas de las válidas

Función independiente
Implementación (automatización con JUnit)

```
60⊕     * La misma clase invalida comprobando la excepcion al estilo JUnit 3
62⊖    @Test
63     public void testClientesNuevosNoPuedenTenerTarjetaJUnit3() {
64         DescuentoModel model=new DescuentoModel();
65         try {
66             model.getDescuento(true, false, true); //caso 4
67             fail("Se deberia producir una excepcion");
68         } catch (RuntimeException e) {
69             assertEquals("Un cliente nuevo no puede disponer de tarjeta de fidelizacior
70         }
71     }
72
74⊕     * Ejemplo de uso de Hamcrest matchers (con assertThat).
80⊖    @Test
81     public void testClientesNuevosHamcrest() {
82         DescuentoModel model=new DescuentoModel();
83         assertThat("caso 1",15,equalTo(model.getDescuento(true, false, false)));
84         assertThat("caso 2",20,equalTo(model.getDescuento(true, true, false)));
85     }
```

Forma clásica de tratar excepciones (desde Junit 3).
Permite probar varias excepciones en mismo método

Forma alternativa de realizar las Comparaciones con Hamcrest Matchers

Función independiente

Implementación (automatización con JUnit: Parámetros)

Para las clases válidas, se trata de pruebas muy repetitivas
(la misma prueba pero con diferente parámetros). Implementación parametrizada.
Usa JUnitParams (https://github.com/Pragmatists/JUnitParams)

- Más simple y compacto que el estandar JUnit4.
- Permite ademas diferentes parametros en la misma clase

```
1  package giis.demo.descuento.ut;
2⊕ import org.junit.*;
10
12⊕ * Pruebas del ejemplo de descuentos de clientes (Problema 3a) usando JUnitPara
24 @RunWith(JUnitParamsRunner.class)
25 public class TestDescuentoFunctionParameters {
27⊕     * Los parametros se pueden especificar en e  e strin
32⊖    @Test
33     @Parameters({
34         "15, true, false, false",
35         "20, true, true, false",
36         "0, false, false, false",
37         "20, false, true, false",
38         "10, false, false, true",
39         "30, false, true, true"})
40     public void testClientesParametrized(int expected, boolean nuevo, boolean cupon, boolean tarjeta) {
41         DescuentoModel model=new DescuentoModel();
42         assertEquals(expected,model.getDescuento(nuevo,cupon,tarjeta));
43     }
45⊕     * Los parametros tambien se pueden especificar en un fichero externo (csv)
47⊖    @Test
48     @FileParameters("src/test/resources/test-parameters.csv")
49     public void testClientesParametrizedFile(int expected, boolean nuevo, boolean cupon, boolean tarjeta) {
50         DescuentoModel model=new DescuentoModel();
51         assertEquals(expected,model.getDescuento(nuevo,cupon,tarjeta));
52     }
```

Requiere especificar el runner a utilizar

Los parámetros se separan Por coma o pipe (|)

Un resultado por cada ejecución (parámetro)

Especificación de parámetros en archivo externo

Runs: 12/12 Errors: 0 Failures: 0

- giis.demo.descuento.ut.TestDescuentoFunctionParameters [Run
  - testClientesParametrized (0,000 s)
    - testClientesParametrized(15, true, false, false) [0] (0,000 s)
    - testClientesParametrized(20, true, true, false) [1] (0,000 s)
    - testClientesParametrized(0, false, false, false) [2] (0,000 s)
    - testClientesParametrized(20, false, true, false) [3] (0,000 s)
    - testClientesParametrized(10, false, false, true) [4] (0,000 s)
    - testClientesParametrized(30, false, true, true) [5] (0,000 s)
  - testClientesParametrizedFile (0,000 s)
    - testClientesParametrizedFile(15, true, false, false) [0] (0,00

Función con Base de Datos
Ejercicio

- Suponer el enunciado anterior ligeramente cambiado:
  - Problema 3b: Se tiene **un informe** que muestra **los clientes en la base de datos a los que se aplica algún** descuento en las compras mediante tarjeta **y el valor de éste**. Los descuentos se determinan como se indica a continuación y son acumulables. Si el cliente acaba de abrir una cuenta de crédito obtiene el 15% de descuento en todas sus compras de hoy, si es un cliente habitual con tarjeta de fidelización obtiene un 10% de descuento. Si el cliente tiene un cupón de descuento obtiene el 20% de descuento (no acumulable con el descuento de nuevo cliente).
- Cuáles serían los casos de prueba

Función con Base de Datos

Implementación (CP)

Ahora:
1 caso de prueba

- Entradas
  - Nuevo Cliente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta fidelización
  - Cliente existente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta de fidelización
- Salidas
  - Acumulación de descuentos
    - Se acumulan
    - No se acumulan
  - Descuento aplicado (%)
    - 15
    - 10
    - 20
    - 0

Entrada (Base de Datos)

ID Cli.	Nuevo	Cupon	Tarjeta
1	S	N	N
2	S	S	N
3	S	N	S
4	N	N	N
5	N	S	N
6	N	N	S
7	N	S	S

Salida (Informe)

ID Cli.	% Dto.
1	15
2	20
5	20
6	10
7	30

Discusión:

En este caso, de las clases de equivalencia se derivan filas, no casos

Función con Base de Datos

Implementación (automatización con JUnit)

```
1  package giis.demo.descuento.ut;
2  import org.junit.*;
8
10  * Pruebas del ejemplo de informe de descuentos de clientes leidos desde la base de datos (Proble
21 public class TestDescuentoDatabase {
22     private static Database db=new Database();
23
24     @BeforeClass
25     public static void setUpClass() {
26         //Aqui se pueden incluir inicializaciones comu
27     }
28     @Before
29     public void setUp() {
30         db.createDatabase(true); //solo la creara la primera vez (para mejorar rendimiento)
31         loadCleanDatabase(db);
32     }
33     @After
34     public void tearDown(){
35         //aqui se cerrarian los objetos ab
36     }
37
39      * Datos de prueba: base de datos  para cubrir las situaciones del dis
43     public static void loadCleanD  se(Database db) {
44         //Otra alternativa  tilizar un script externo con las queries y ejecuta
45         //(en este caso se pondria en src/test/resources)
46         db.executeBatch(new String[] {
47                 "delete from clientes",
48                 "insert into clientes(id,edad,nuevo,cupon,tarjeta) values (1,18,'S','N','N')",
49                 "insert into clientes(id,edad,nuevo,cupon,tarjeta) values (2,38,'S','S','N')",
50                 "insert into clientes(id,edad,nuevo,cupon,tarjeta) values (3,21,'S','N','S')",
51                 "insert into clientes(id,edad,nuevo,cupon,tarjeta) values (4,25,'N','N','N')",
52                 "insert into clientes(id,edad,nuevo,cupon,tarjeta) values (5,40,'N','S','N')",
53                 "insert into clientes(id,edad,nuevo,cupon,tarjeta) values (6,42,'N','N','S')",
54                 "insert into clientes(id,edad,nuevo,cupon,tarjeta) values (7,39,'N','S','S')",
55             });
56     }
```

Normalmente se usará una BD común para los tests de cada clase. Asegura que cada método se ejecuta con una BD limpia en un estado perfectamente conocido.
Recordar que cada los test deben ser independientes, ejecutables en cualquier orden

Crear métodos reusables para reducir código,
Eliminar complejidades de conexiones, excepciones…
Centrarse en probar, no en programar de más

Cuando hay muchas tablas, las SQL
Mejor en un archivo externo en
src/test/resources

Función con Base de Datos

Implementación (automatización con JUnit)

En este ejercicio solo había un caso de prueba

```
59⊕     * Para la consulta sin parametros simplemente invoca el metodo del modelo que obtiene una lista de objeto
63⊖    @Test
64     public void testConsultaSinParametro() {
65         DescuentoModel model=new DescuentoModel();
66         List<DescuentoDisplayDTO> descuentos=model.getListaDescuentos();
67         assertEquals("1,15\n"
68                 +"2,20\n"
69                 +"5,20\n"
70                 +"6,10\n"
71                 +"7,30\n",
72                 Util.pojosToCsv(descuentos,new String[] {"id","descuento"}));
73     }
74⊖    /**
```

Simplificar la comparación de resultados.
Intentar que sea fácil de modificar y comparar
En este caso compara entre textos CSV
(Eclipse permite ver fácilmente las diferencias)

Otro método reusable que permitirá comparación fácil
de los resultados y disminuir el código de prueba
(en este caso usando Apache Commons BeanUtils)

Función con Base de Datos y parámetros
Ejercicio

- Suponer el enunciado anterior ligeramente cambiado:
  - Problema 3c: Se tiene un informe que muestra los clientes en la base de datos a los que se aplica algún descuento en las compras mediante tarjeta y el valor de éste. Los descuentos se determinan como se indica a continuación y son acumulables. Si el cliente acaba de abrir una cuenta de crédito obtiene el 15% de descuento en todas sus compras de hoy, si es un cliente habitual con tarjeta de fidelización obtiene un 10% de descuento. Si el cliente tiene un cupón de descuento obtiene el 20% de descuento (no acumulable con el descuento de nuevo cliente). **El informe tiene un parámetro opcional (edad) que si está presente, oculta los resultados de aquellos con edad menor que la especificada**
- Cómo cambia el diseño de las pruebas?

Función con Base de Datos y parámetros

Implementación (CP)

- Entradas
  - Nuevo Cliente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta fidelización (inválida)
  - Cliente existente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta de fidelización
- Salidas
  - Acumulación de descuentos
    - Se acumulan
    - No se acumulan
  - Descuento aplicado (%)
    - 15
    - 10
    - 20
    - 0

- Entradas adicionales
  - Filtrado por edad
    - No especificado
    - Cliente filtrado (en VL)
    - Cliente no filtrado (en VL)

(Base de Datos)

ID Cli.	Edad	Nuevo	Cupon	Tarjeta
1	18	S	N	N
2	38	S	S	N
3	21	S	N	S
4	25	N	N	N
5	40	N	S	N
6	42	N	N	S
7	39	N	S	S

CP	Edad
1	No espec.
2	40

(Salidas)

ID Cli.	% Dto.
1	15
2	20
5	20
6	10
7	30

ID Cli.	% Dto.
5	20
6	10

Datos de prueba en Bases de datos

- Cuando el programa procesa datos en una base de datos, los datos almacenados en ésta también de deben considerar como entrada (y salida)
  - Las situaciones a probar se representan:
    - Como filas en tablas de la base de datos
    - Como entradas adicionales (p.e. parámetros) -> Caso de Prueba
    - Varios casos de prueba pueden compartir la misma base de datos
  - Con frecuencia
    - Muchas veces nos referimos a los datos de la base de datos como Datos de Prueba (Test Data), formando parte del estado inicial de las pruebas
    - Se reserva la palabra entrada para las entradas adicionales
- Idem. para las salidas

Discusión:

Probar con muchos/pocos datos

Cómo insertar los datos

Función con Base de Datos

Implementación (automatización con JUnit)

En este ejercicio hay dos casos de prueba (el primero ya se había implementado)

```
59⊕     * Para la consulta sin parametros simplemente invoca el metodo del modelo que c
63⊖    @Test
64     public void testConsultaSinParametro() {
65         DescuentoModel model=new DescuentoModel();
66         List<DescuentoDisplayDTO> descuentos=model.getListaDescuentos();
67         assertEquals("1,15\n"
68                 +"2,20\n"
69                 +"5,20\n"
70                 +"6,10\n"
71                 +"7,30\n",
72                 Util.pojosToCsv(descuentos,new String[] {"id","descuento"}));
73     }
74⊖    /**
75      * La misma forma de probar cuando hay parametros.
76      */
77⊖    @Test
78     public void testConsultaConParametro() {
79         DescuentoModel model=new DescuentoModel();
80         List<DescuentoDisplayDTO> descuentos=model.getListaDescuentos(40);
81         assertEquals("5,20\n"
82                 +"6,10\n",
83                 Util.pojosToCsv(descuentos,new String[] {"id","descuento"}));
84     }
```

Se refiere a parámetros de la función a probar,
No a la parametrización del test

Si hubiera más casos además de este (40)
se parametrizaría la prueba
(ver ejemplos con Spring)
El CSV de la salida esperada sería
Un parámetro más

… Herramientas para Prácticas

Datos de entrada y ejecución desde SQLTest

Diseñar Pruebas

Programa

Q1

RUN

Base de datos inicial

BD01

RUN

Add

Del

Ren

Caso de prueba

SinFiltro

ConFiltro

RUN

Add

Del

Ren

Clientes

```
1 18 S N N
2 38 S S N
3 21 S N S
4 25 N N N
5 40 N S N
6 42 N N S
7 39 N S S
```

SQL a ejecutar antes del caso

Entradas de usuario

40

Salida deseada

```
5 20
6 10
```

Salida obtenida

```
5 20
6 10
```

Resultados y Mensajes de Error

```
(+) OK - Test: /testdescuento/tuya/Q1/BD01/SinFiltro
(+) OK - Test: /testdescuento/tuya/Q1/BD01/ConFiltro
------- Resumen de casos ejecutados -------
2 (+) validacion OK
0 (-) validacion NO OK
0 (?) errores
```

Comandos SQL Eje

```
--Test: /testdescue
--SQL: select id, d
(select id, edad, C
WHEN nuevo='S' and
WHEN nuevo='S' and
WHEN nuevo='N' and
```

… Herramientas para Prácticas

Diseño de las pruebas (TestCel)

RP				id	Descripción	trr	trr
1			^	**1**	**Datos del Cliente**		
2			^	**1.1**	**Nuevo Cliente**		
3				1.1.1	Sin cupón descuento ni tarjeta fideliz.	1	□
4				1.1.2	Con cupón descuento	1	□
5				1.1.3	Con tarjeta fidelización (inválida)	1	□
6			^	**1.2**	**Cliente existente**		
7				1.2.1	Sin cupón descuento ni tarjeta fideliz.	1	□
8				1.2.2	Con cupón descuento	1	□
9				1.2.3	Con tarjeta de fidelización	1	□
10			^	**2**	**Filtrado por edad**		
11				2.1	No especificado	1	□
12				2.2	Cliente filtrado (en VL)	1	✓
13				2.3	Cliente no filtrado (en VL)	1	✓
14			^	**3**	**Acumulación de descuentos (salidas)**		
15				3.1	Se acumulan	1	□
16				3.2	No se acumulan	1	□
17			^	**4**	**Descuento aplicado (%) (salidas)**		
18				4.1	15	1	□
19				4.2	10	1	□
20				4.3	20	1	□
21				4.4	0	1	□
22							

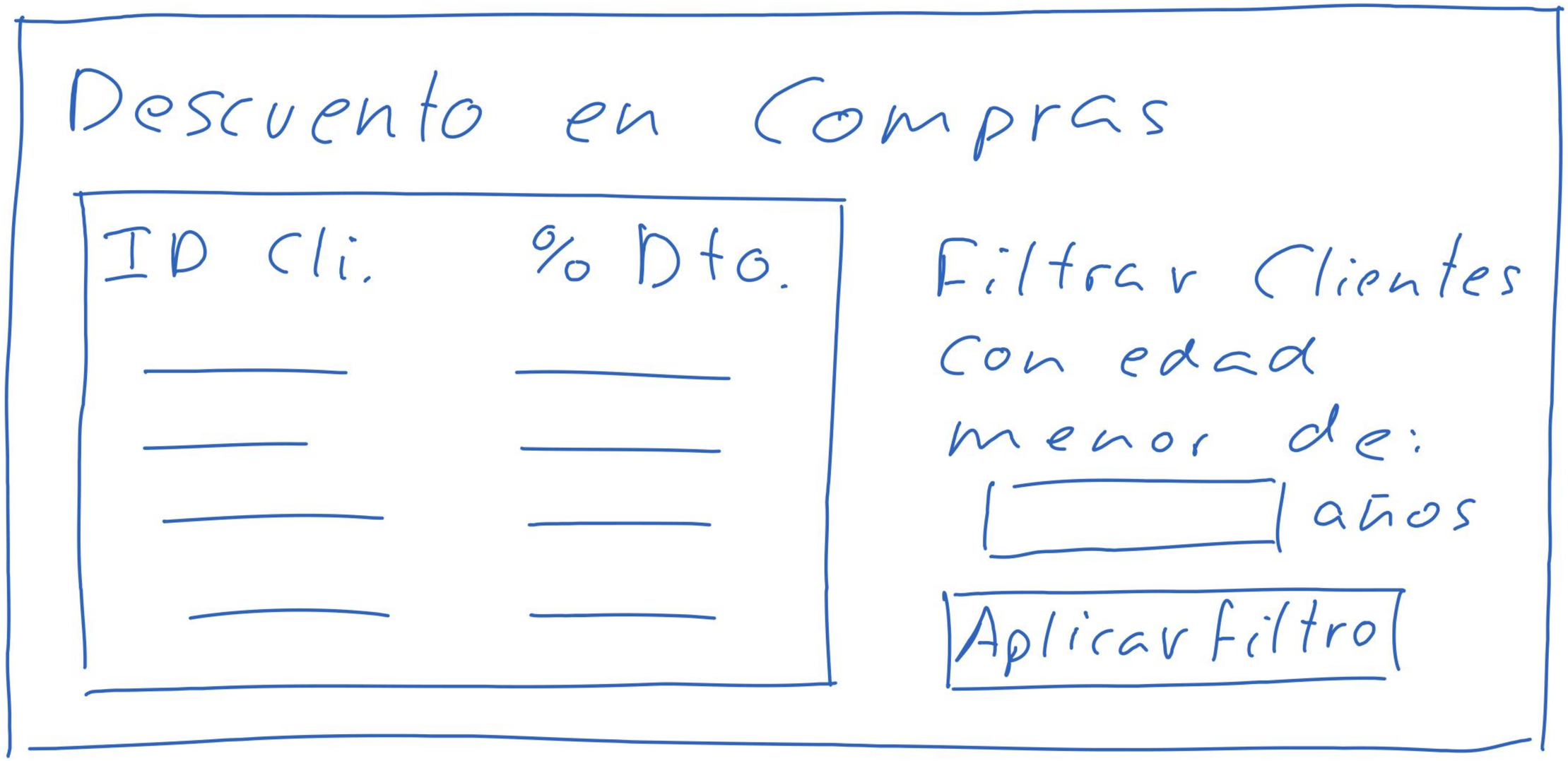
CP			trl	trl	id	Descripción	Cond. Inicial	Procedimient	Salida Esperada	Estado
1				13	1	Informe sin filtros	BD 01 conteniendo todas las situaciones relativas a Datos del Cliente	Ejecutar sin parámetros	Todos los registros excepto situaciones imposibles o descuentos cero (se filtran IDS 3 y 4)	Pasa
2	◄	►	⇆	2	2	Informe con filtro por edad	idem	Ejecutar filtrando por 40 años	Se filtra todo excepto IDs 5 y 20	Pasa
3										

Prueba del interfaz de usuario
Ejercicio

- Suponer el enunciado anterior ligeramente cambiado:
  - Problema 3d: Se tiene un informe que muestra los clientes en la base de datos a los que se aplica algún descuento en las compras mediante tarjeta y el valor de éste. Los descuentos se determinan como se indica a continuación y son acumulables. Si el cliente acaba de abrir una cuenta de crédito obtiene el 15% de descuento en todas sus compras de hoy, si es un cliente habitual con tarjeta de fidelización obtiene un 10% de descuento. Si el cliente tiene un cupón de descuento obtiene el 20% de descuento (no acumulable con el descuento de nuevo cliente). El informe tiene un parámetro opcional (edad) que si está presente, oculta los resultados de aquellos con edad menor que la especificada. **Este parámetro es indicado desde la pantalla del usuario (se supone que ya está validado el formulario).**

Prueba del interfaz de usuario

Ejercicio



- Cómo cambia el diseño de las pruebas?

Prueba del interfaz de usuario

Diseño e Implementación

- Entradas
  - Nuevo Cliente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta fidelización (inválida)
  - Cliente existente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta de fidelización
  - Filtrado por edad
    - No especificado
    - Cliente filtrado (en VL)
    - Cliente no filtrado (en VL)
- Salidas
  - Acumulación de descuentos
    - Se acumulan
    - No se acumulan
  - Descuento aplicado (%)
    - 15
    - 10
    - 20
    - 0
- Acciones Usuario
  - Cambio filtro edad
    - Se pone
    - Se cambia valor
    - Se quita

(BD *)

ID Cli.	Edad	Nuevo	Cupon	Tarjeta
1	18	S	N	N
2	38	S	S	N
3	21	S	N	S
4	25	N	N	N
5	40	N	S	N
6	42	N	N	S
7	39	N	S	S

(Salida *)

ID Cli.	% Dto.
1	15
2	20
5	20
6	10
7	30

	Inicial	Entrada/ procedim.	Salida Esperada
1	BD *	Solo abrir app.	Muestra todo (Salida *)
2	BD *	Incluir filtro a 40	Muestra ID 5,6
3	BD *	Pasar filtro de 40 a 39	Muestra ID 5,6,7
4	BD *	quitar filtro	Muestra todo (Salida *)

Prueba del interfaz de usuario
Diseño e Implementación

- Cuando hay interfaz de usuario:
  - Probar las acciones que realiza
  - Que causan CAMBIOS en lo visualizado
  - Y posiblemente en la base de datos
- Muchas veces esto da lugar a casos de prueba compuestos de una serie de PASOS relacionados unos con otros.
- Script para ejecución manual:

	Descripción /Objetivo	Cond. Inicial	Entrada /procedimiento	Salida Esperada
1	Efecto de los filtros por edad	BD *	(1) Abrir app (2) filtrar a 40 (3) filtrar a 39 (4) quitar filtro	(1) Muestra todo (Salida *) (2) deja solo ID 5,6 (3) añade ID 7 (4) vuelve a mostrar todo

- No abusar con casos de prueba con excesivo número de pasos

Prueba del interfaz de usuario

Implementación (automatización con Junit y AssertJ)

Necesitamos un componente para simular la interacción del usuario con la aplicación. Se usa p.e. **AssertJ Swing** (análogo a Selenium)

```
1  package giis.demo.descuento.it;
2⊕ import org.assertj.swing.fixture.FrameFixture;
8
10⊕ * Pruebas de la interaccion del usuario con la aplicacion swing del ejemplo de descuentos a clientes
24 public class ITDescuento {
25     private FrameFixture window; //ventana que esta siendo objeto de prueba en cada momento
26
27     private Database db=new Database();
28     @BeforeClass
29       public static void setUpOnce() {
30         FailOnThreadViolationRepaintManager.install();
31       }
32     @Before
33     public void setUp() {
34         db.createDatabase(true);
35         //Utiliza el mismo setup de datos que en los tests ut
36         giis.demo.descuento.ut.TestDescuentoDatabase.loadCleanDatabase(db);
37         //Lanza main y selecciona la opcion para abrir la ventana bajo prueba (ver implementacion de e
38         window=AssertjUtil.getApplicationFixture("Ejecutar giis.demo.descuento", "Descuento");
39     }
40     @After
41     public void tearDown(){
42         window.cleanUp();
43     }
```

Cada método se ejecutará con una BD limpia como antes y después se abre la pantalla de la aplicación a probar. (Se ha creado otro método reutilizable para ello, ver código)

Necesaria la limpieza aquí (cerrar ventana). Imporante: al ejecutar la prueba AssertJ Swing toma control del interfaz (no interferir en el UI mientas se ejecuta)

Prueba del interfaz de usuario

Implementación (automatización con Junit y AssertJ)

El caso de prueba es
un ESCENARIO con 4 pasos
(aquí no se puede separar
en 4 métodos independientes)

Verificación de valor inicial de la pantalla.
Hay métodos para localizar componentes
del UI por texto, nombre, etc, y realizar
asserts con los matchers propios

```
44    /*
45     *
46     *                                   d, comp                                   o
47     */
48    @Test
49    public void testDescuentoScenario() {
50        //Paso1: estado inicial, no hay seleccion.
51        window.textBox("txtAnyos").requireText("");
52        assertEquals("1,15\n2,20\n5,20\n6,10\n7,30\n",
53                Util.arraysToCsv(window.table("tabDescuentos").contents()));
54        AssertjUtil.takeScreenshot(window,"descuentoPaso1");
55        //Paso2: incluir filtro a 40
56        window.textBox("txtAnyos").setText("40");
57        window.button("btnAplicarFiltro").click();
58        assertEquals("5,20\n6,10\n",
59                Util.arraysToCsv(window.table("tabDescuentos").contents()));
60        AssertjUtil.takeScreenshot(window,"descuentoPaso2");
61        //Paso3: pasar filtrro de 40 a 39
62        window.textBox("txtAnyos").setText("39");
63        window.button("btnAplicarFiltro").click();
64        assertEquals("5,20\n6,10\n7,30\n",
65                Util.arraysToCsv(window.table("tabDescuentos").contents()));
66        AssertjUtil.takeScreenshot(window,"descuentoPaso3");
67        //Paso4: quitar filtro
68        window.textBox("txtAnyos").setText("");
69        window.button("btnAplicarFiltro").click();
70        assertEquals("1,15\n2,20\n5,20\n6,10\n7,30\n",
71                Util.arraysToCsv(window.table("tabDescuentos").contents()));
72        AssertjUtil.takeScreenshot(window,"descuentoPaso4");
73    }
```

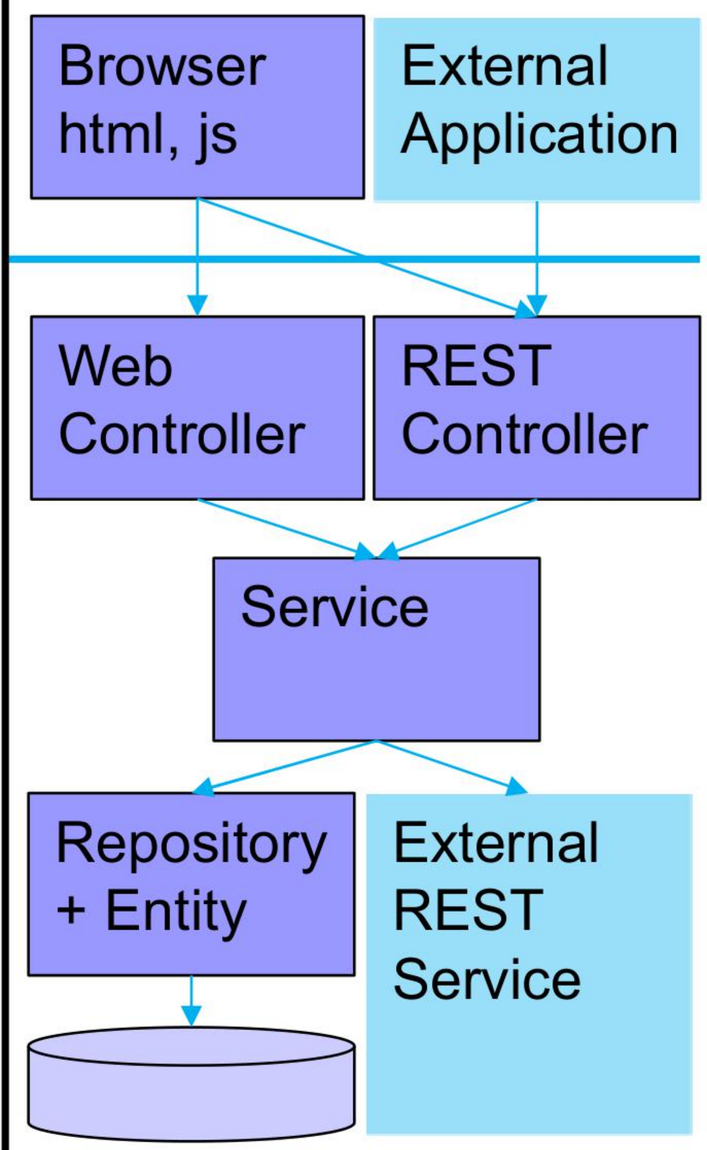
Idem, pero sobre el contenido
de una tabla.
Utiliza otro método reutilizable para
Obtener esta tabla en un string csv

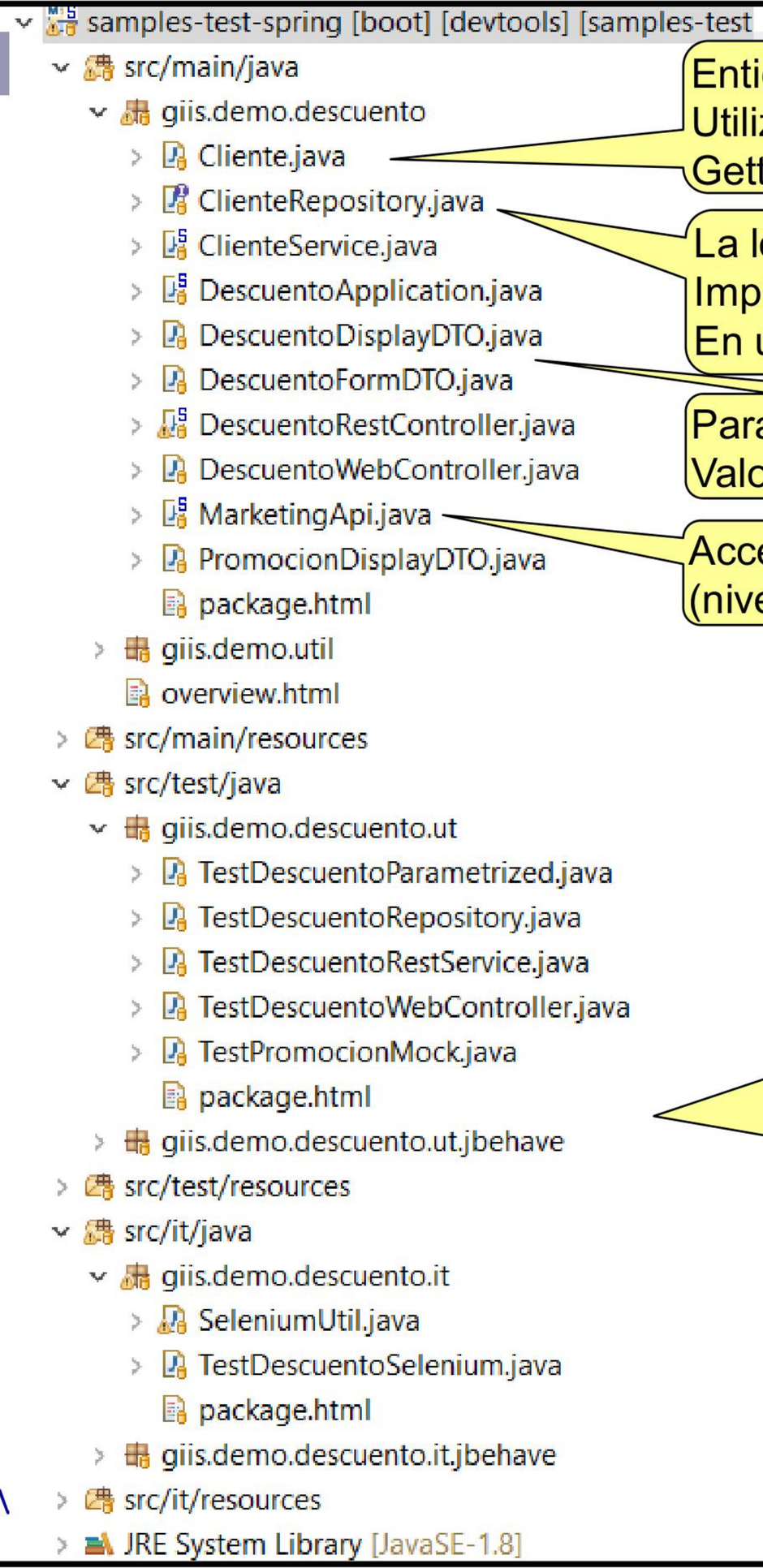
Obtiene el estado intermedio
De la ventana en una imagen
Para depuración

En el resto de pasos se realiza la
Acción (setText) y comparaciones
similares a las anteriores

Demasiado código?
Pensar en encapsular cada paso
(se hará en las pruebas con Spring)

Automatización con Spring Boot





Entidad representando la BD
Utiliza Lombok para los
Getters y setters

La lógica de negocio se
Implementa con una SQL
En una anotación del repo.

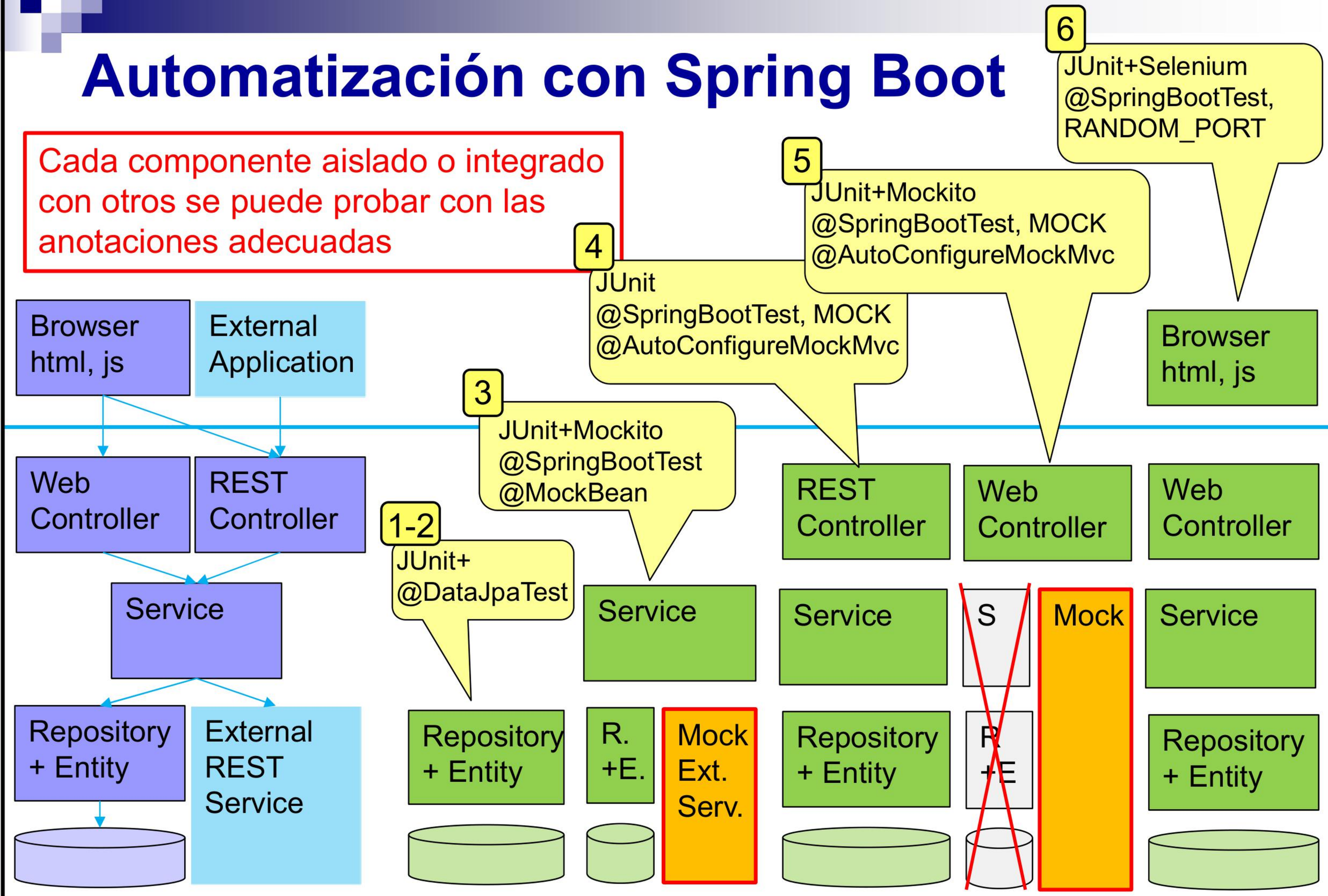
Parametros del formulario y
Valores devueltos en request

Acceso a api REST externa
(nivel de servicio)

Todos los tests
Realizados en
Diferente niveles
Y con diferentes
Niveles de
Integración de
Los componentes

Automatización con Spring Boot

Cada componente aislado o integrado con otros se puede probar con las anotaciones adecuadas



1-Prueba unitaria del Repositorio

```
1  package giis.demo.descuento.ut;
2⊕ import org.junit.*;
17
19⊕ * Pruebas del ejemplo de informe de descuentos de clientes leidos desde
32 @DataJpaTest
33 @TestPropertySource(locations="classpath:application-test.properties")
34 @RunWith(SpringRunner.class)
35 public class TestDescuentoRepository {
36     //para cargar datos de prueba
37     @Autowired private TestEntityManager entityManager;
38     //el repositorio bajo prueba
39     @Autowired private ClienteRepository cliente;
40     //datasource para acceso a la base de datos mediante sql con JdbcTemplate
41     @Autowired private javax.sql.DataSource datasource;
42
43⊖    @Before
44     public void setUp() {
45         loadCleanDatabase();
46     }
48⊕     * Datos de prueba que se cargaran en el setup para cubrir las si
53⊖    public void loadCleanDatabase() {
54         //datos cargados a traves del TestEntityManager
55         entityManager.persist(new Cliente(1,18,"S","N","N"));
56         entityManager.persist(new Cliente(2,38,"S","S","N"));
57         entityManager.persist(new Cliente(3,21,"S","N","S"));
58         //datos cargados directamente a traves del repositorio
59         cliente.save(new Cliente(4,25,"N","N","N"));
60         cliente.save(new Cliente(5,40,"N","S","N"));
61         //datos cargados directamente en la base de datos utilizando sql
62         JdbcTemplate database=new JdbcTemplate(datasource);
63         database.execute("insert into cliente(id,edad,nuevo,cupon,tarjeta) values"
64                 +"(6,42,'N','N','S'),"
65                 +"(7,39,'N','S','S')");
66     }
```

Permite acceder a las entidades de la base de datos con una configuración especifica para pruebas y BD en memoria

Configuracion especifica para estas pruebas (omite la carga de datos inicial de data.sql)

Requiere runner específico de Spring

Tres objetos diferentes para ilustrar tres formas Diferentes de cargar los datos:

- Métodos de DataJpaTest
- Acceso directo al respositorio
- Acceso directo a la base de datos jdbc

1-Prueba unitaria del Repositorio

Los métodos de prueba iguales que en samples-test-java

```
69⊕     * Para la consulta sin parametros simplemente invoca el metodo del modelo que obtie
73⊖    @Test
74     public void testConsultaSinParametro() {
75         List<DescuentoDisplayDTO> descuentos=cliente.getListaDescuentos(null);
76         assertEquals("1,15\n"
77                 +"2,20\n"
78                 +"5,20\n"
79                 +"6,10\n"
80                 +"7,30\n",
81                 Util.pojosToCsv(descuentos,new String[] {"id","descuento"}));
82     }
84⊕     * La misma forma de probar cuando hay parametros.
86⊖    @Test
87     public void testConsultaConParametro() {
88         List<DescuentoDisplayDTO> descuentos=cliente.getListaDescuentos(40);
89         assertEquals("5,20\n"
90                 +"6,10\n",
91                 Util.pojosToCsv(descuentos,new String[] {"id","descuento"}));
92     }
```

2-Prueba unitaria del Repositorio
Parametrizada

```
1  package giis.demo.descuento.ut;
2⊕ import org.junit.*;
19
21⊕ * Pruebas del ejemplo de informe de descuentos de clientes leidos desde la base c
27 @DataJpaTest
28 @TestPropertySource(locations="classpath:application-test.properties")
29 //En este caso no se utiliza SpringRunner
30 @RunWith(JUnitParamsRunner.class)
31 public class TestDescuentoParametrized {
32     @Autowired private TestEntityManager entityManager;
33     @Autowired private ClienteRepository cliente;
34
35     //Para sustituir SpringRunner basta con definir estas dos reglas
36     //(solo disponibles en ultimas versiones de spring)
37     @ClassRule
38     public static final SpringClassRule SPRING_CLASS_RULE = new SpringClassRule();
39     @Rule
40     public final SpringMethodRule springMethodRule = new SpringMethodRule();
41
42     @Before
43     public void setUp() {
44         loadCleanDatabase();
45     }
```

No se usa el runner de Spring, sino el de la prueba parametrizada (con JUnitParams)

Pero el runner de Spring es necesario para establecer la configuración Necesaria para las pruebas.
Estas dos reglas causan el mismo efecto Cuando no se utiliza el runner de Spring

2-Prueba unitaria del Repositorio
Parametrizada

```
47    public void loadCleanDatabase() {
48        entityManager.persist(new Cliente(1,18,"S","N","N"));
49        entityManager.persist(new Cliente(2,38,"S","S","N"));
50        entityManager.persist(new Cliente(3,21,"S","N","S"));
51        entityManager.persist(new Cliente(4,25,"N","N","N"));
52        entityManager.persist(new Cliente(5,40,"N","S","N"));
53        entityManager.persist(new Cliente(6,42,"N","N","S"));
54        entityManager.persist(new Cliente(7,39,"N","S","S"));
55    }
56
58     * El test parametrizado ejecuta el proceso de obtencion de los
61    @Test
62    @Parameters({"39, 5;20\n6;10\n7;30\n",
63                 "40, 5;20\n6;10\n"})
64    public void testParametrizado(Integer edad, String expected) {
65        List<DescuentoDisplayDTO> descuentos=cliente.getListaDescuentos(edad);
66        assertEquals(expected.replace(";", ","),
67                Util.pojosToCsv(descuentos,new String[] {"id","descuento"}).trim());
68    }
```

Los mismos datos que en las anteriores

Los valores esperados obtenidos de la BD
Los definimos como csv, pero con JUnitParams el separador (coma) esta reservado para los parámetros.
Uso otro separador para ellos

Un pequeño postprocesamiento:
- vuelve a poner coma en expected como separador
- elimina salto de línea final en actual

(se podría adaptar el método de utilidad para evitar esto)

3-Prueba unitaria con Mocks

Ejercicio adicional

- Para esto supondremos otra funcionalidad adicional:
  - Mostrar la lista de clientes que tienen alguna promoción aplicable, junto con el código de la promoción.
    - Las promociones dependen del país (atributo del cliente).
    - Los códigos de promociones aplicables a cada país se almacenan y gestionan en un microservicio externo al que se accede mediante un api REST
- Dos supuestos típicos:
  - El servicio externo todavía **no está implementado** todavía
  - Como parte de la funcionalidad está en un servicio externo, queremos independizarnos de éste y **probar la lógica de negocio de forma unitaria**

3-Prueba unitaria con Mocks

Diseño e implementación

- Situaciones a cubrir
  - Codigos de promocion del cliente segun pais
    - tiene codigo de promocion
    - no tiene codigo de promocion
    - hay codigos de promocion diferentes

1 caso de prueba

Entrada (Base de Datos)

ID Cli.	Nuevo	Cupon	Tarjeta	País
1	S	N	N	ES
2	N	S	N	US
3	N	N	S	UK

Entrada (Microservicio)

Pais	ID Promo
ES	P01ES
UK	P03UK

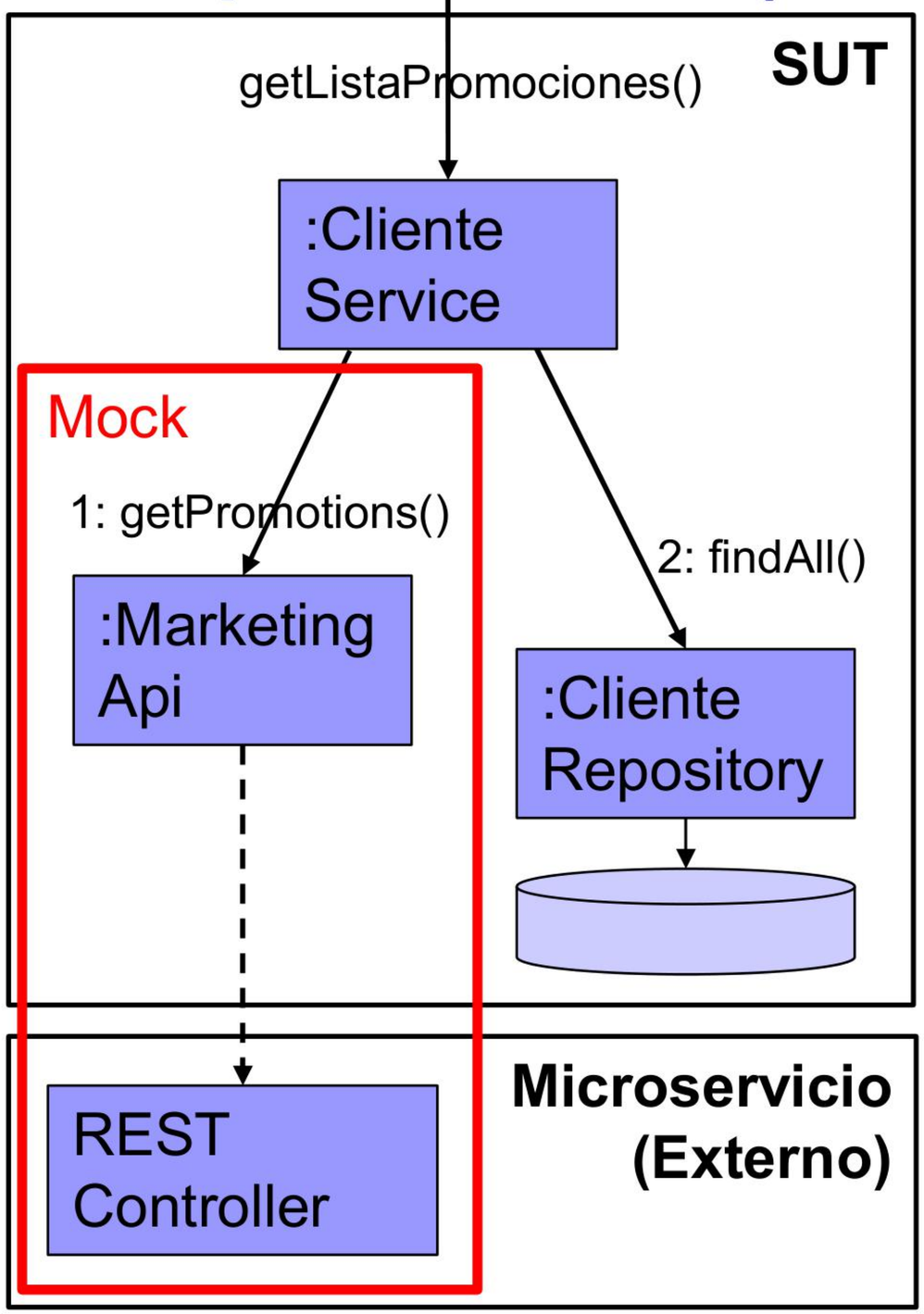
Salida (Informe)

ID Cli.	ID Promo
1	P01ES
2	P03UK

- Las situaciones a cubrir y los casos de prueba son iguales que si tuviésemos todos los datos en la misma BD
- Cambiará el script (la forma de ejecutar/automatizar)

3-Prueba unitaria con Mocks

Implementación (automatización)



- Arquitectura:
  - La lógica de negocio está implementada en la capa de servicio
  - El sistema a probar tiene una clase (a nivel de servicio) que define un api cliente para acceder al microservicio externo
  - Busca los códigos de promoción y país
  - Asocia a cada cliente el código de promoción según su país, devolviendo un DTO
- El Mock remplaza el api cliente (no implementado todavía)
  - Nota: En la práctica se usa el término Mock para referirse a diferentes conceptos que siendo estrictos, son diferentes (mock, stub, fake…): https://martinfowler.com/articles/mocksArentStubs.html

3-Prueba unitaria con Mocks

Implementación (automatización)

Estoy probando un servicio y establezco la configuración de la aplicación

```
1  package giis.demo.descuento.
2⊕ import org.junit.*;
20
22⊕ * Prueba de un servicio con una implementacion incompleta: Ilustra el uso de Mocks y la carga
37 @SpringBootTest(classes = DescuentoApplication.class)
38 @TestPropertySource(locations="classpath:application-test.properties")
39 @RunWith(SpringRunner.class)
40 public class TestPromocionMock {
41     //El servicio bajo prueba
42     @Autowired private ClienteService cliente;
43     //el mock que sustituira los metodos que acceden al microservicio
44     //que utiliza ClienteService (no implementados todavia)
45     @MockBean private MarketingApi marketing;
46
48⊕     * Define el mock que devuelve una lista de pares clave-valor de la forma codigo de pais-codigo de promocio
51⊖    @SuppressWarnings("serial")
52     @Before
53     public void setUp() {
54         Map<String, String> codes = new HashMap<String, String>() {{ put("ES", "P01ES"); put("UK", "P03UK"); }}
55         Mockito.when(marketing.getPromotions()).thenReturn(codes);
56     }
```

En vez de Autowired, MockBean indica Que no se usará la clase real, sino un mock, Spring Boot integra automáticamente Mockito

El mock es construido con un conjunto de reglas Que producen una salida cuando se invoca un método

En este caso el Map que contiene los datos de prueba Diseñados, podría haber mas reglas y parámetros En los metodos

3-Prueba unitaria con Mocks

Implementación (a

Ilustra otra forma de cargar los datos:
Ejecutando un script externo con las SQL que se ejecutaran,
El archivo se sitúa en src/test/resources

```
59⊕     * El caso de prueba utiliza el servicio del client como si toda la implementacion de la api
66⊖    @Sql(scripts = "classpath:/sql-test-mock.sql",
67             config = @SqlConfig(commentPrefix = "--", separator = ";")) //config se podria quitar pu
68     @Test
69     public void testPromocionMock() {
70         List<PromocionDisplayDTO> promos=cliente.getListaPromociones();
71         assertEquals("1,P01ES\n2,\n3,P03UK\n",Util.pojosToCsv(promos,new String[] {"id","promo"}));
72     }
```

El método de prueba es igual que
Si no se utilizasen mocks

La implementación del servicio no ha cambiado, simplemente
Se ha sustituido esta implementación del objeto marketing por el mock

```
public List<PromocionDisplayDTO> getListaPromociones() {
    //datos que seran fusionados (uno procede del repositorio y otro del a
    Map<String,String> promosPorPais=marketing.getPromotions();
    List<Cliente> clientes=cliente.findAll();
    //forma la lista de promociones
    List <PromocionDisplayDTO> clientesConPromo=new ArrayList<>();
    for (Cliente item : clientes) {
        String promo=promosPorPais.get(item.getPais());
        clientesConPromo.add(new PromocionDisplayDTO(item.getId(),promo));
    }
    return clientesConPromo;
}
```

4-Prueba de un servicio REST

```
1  package giis.demo.descuento.ut;
2⊕ import org.junit.*;
22
24⊕ * Pruebas del ejemplo de informe de descuentos de client          desde la base de dat
39 @SpringBootTest(classes={DescuentoApplication.class},
40     webEnvironment=SpringBootTest.WebEnvironment.MOCK)
41 @AutoConfigureMockMvc
42 @TestPropertySource(locations="classpath:application-test.properties")
43 @RunWith(SpringRunner.class)
44 public class TestDescuentoRestService {
45     //datasource para acceso a la base de datos mediante sql con JdbcTemplate
46     @Autowired private javax.sql.DataSource datasource;
47     //Objeto usado para acceder al servicio rest
48     @Autowired private MockMvc mvc;
49
50     @Before
51     public void setUp() {
52         loadCleanDatabase();
53     }
54     public void loadCleanDatabase() {
55         JdbcTemplate database=new JdbcTemplate(datasource);
56         database.execute("delete from cliente");
57         database.execute("insert into cliente(id,edad,nuevo,cupon,tarjeta) values"
58                 +"(1,18,'S','N','N'),"
59                 +"(2,38,'S','S','N'),"
60                 +"(3,21,'S','N','S'),"
61                 +"(4,25,'N','N','N'),"
62                 +"(5,40,'N','S','N'),"
63                 +"(6,42,'N','N','S'),"
64                 +"(7,39,'N','S','S')");
65     }
```

Establece la configuración como aplicación web, pero sin desplegar el servidor Tomcat

configura MockMvc que permite acceder a los endpoints del servicio.

El mock del cliente que accederá al servicio REST

La carga de datos como siempre, en este Caso con JdbcTemplate para cargar directamente la base de datos

4-Prueba de un servicio REST

```
71  @Test
72  public void testConsultaSinParametro() throws Exception {
73      //el objeto mvc ejecuta la llamada al api y contiene una serie d  chers para comp
74      //el resultado (se muestran diferentes formas de comprobar)
75      mvc.perform(get("/api/descuentos").contentType(MediaType.APPLICATION_JSON))
76              .andExpect(status().isOk())
77              .andExpect(jsonPath("$", hasSize(5)))
78              .andExpect(jsonPath("$[0].id", is(1)))
79              .andExpect(jsonPath("$[0].descuento", is(15)))
80              .andExpect(jsonPath("$[1].*",contains(2,20)))
81              .andExpect(jsonPath("$[2].*",contains(5,20)))
82              .andExpect(jsonPath("$[3].*",contains(6,10)))
83              .andExpect(jsonPath("$[4].*",contains(7,30)));
84  }
85  @Test
86  public void testConsultaConParametro() throws Exception {
87      //El objeto mvc devuelve un ResultActions que puede ser utilizado pa
88      ResultActions x=mvc.perform(get("/api/descuentos?edad=40")
89              .contentType(MediaType.APPLICATION_JSON))
90              .andExpect(status().isOk());
91      //en este caso se comparara el contenido completo del json obtenido
92      String json=x.andReturn().getResponse().getContentAsString();
93      assertEquals("[{\"id\":5,\"descuento\":20},{\"id\":6,\"descuento\":10}]",json);
94      //ahora eliminando comillas para que sea mas facil indicar la salida deseada
95      assertEquals("[{id:5,descuento:20},{id:6,descuento:10}]",json.replaceAll("\"", ""));
96  }
```

Para la prueba se accede directamente a la Dirección del api, indicando que los datos son json

Ilustra diferentes formas de comprobar los datos Devueltos:

- Status devuelto (200 OK)
- Comprobaciones de tamaño
- Comprobaciones de contenido en fragmentos

Otra prueba, comprobando solamente que El status es OK

Aquí en la comprobación de la Salida, comparo directamente El json devuelto

Aquí hago una pequeña transformación para facilitar la escritura de la salida

5-Prueba unitaria de un controlador web

Tenemos controladores REST y Web,
y pruebas con y sin mocks.
Antes Rest sin mocks, ahora Web con mocks

Probar controlador de forma independiente si tiene
Lógica compleja que no se probará desde el UI
(si no se probaría lo mismo dos veces)

Configuración similar al probar controlador REST, salvo:

- No especifica contexto (solo se prueba el controlador)
- No especifica TestPropertySource (se prueba sin BD)

El mock devolvera los datos de ClienteService (id,descuento) simulando el comportamiento con una base de datos con dos filas (1,18,'S','N','N'), (2,38,'S','S','N') para dos situaciones:

1. no edad (mostrar todas las filas, descuentos 15, 20)
2. edad 19 (mostrar solo segunda fila, descuento 20)

```
1  package giis.demo.descuento.ut;
2⊕ import org.junit.*;
27
29⊕  * Ilustra la configuracion para pruebas unitarias de
40 @SpringBootTest
41 @AutoConfigureMockMvc
42 @RunWith(SpringRunner.class)
43 public class TestDescuentoWebController {
44     //Objeto usado para acceder al controlador web
45     @Autowired private MockMvc mvc;
46     //el mock del servicio que utiliza este controlado
47     @MockBean private ClienteService cliente;
48
50⊕     * Configura el mock en el setup: En este ejemplo
56⊖    @SuppressWarnings("serial")
57     @Before
58     public void setUp() {
59         DescuentoDisplayDTO descuento0=new DescuentoDisplayDTO(1,15);
60         DescuentoDisplayDTO descuento1=new DescuentoDisplayDTO(2,20);
61         Mockito.when(cliente.getListaDescuentos(0))
62             .thenReturn(new ArrayList<DescuentoDisplayDTO>() {{ add(descuento0); add(descuento1); }});
63         Mockito.when(cliente.getListaDescuentos(19))
64             .thenReturn(new ArrayList<DescuentoDisplayDTO>() {{ add(descuento1); }});
65     }
```

5-Prueba unitaria de un controlador web

Como en un servicio REST se utiliza mvc para invocar el controlador con un GET

El controlador se comunica con la vista con dos objetos:
"command“: campo donde el usuario introduce la edad
"descuento“: lista de descuentos devueltos

```
68⊕     * Situación (1): Get para obtener la pagina inicial, no se especifica edad, devolvera dos filas (de acuerdo con el mock).
73⊖    @Test
74     public void testGetRequest() throws Exception {
75         ResultActions res=mvc.perform(get("/descuentos"))
76                 .andExpect(status().isOk());
77         //la comprobacion se realiza accediendo los DTOs que el controlador recibe/devuelve en el request (Model)
78         res.andExpect(model().attribute("command", hasProperty("edad", is(nullValue()))));
79         res.andExpect(model().attribute("descuentos", hasSize(2)));
80         res.andExpect(model().attribute("descuentos", hasItem(
81                 allOf(hasProperty("id", is(1)), hasProperty("descuento", is(15)) ) )));
82         res.andExpect(model().attribute("descuentos", hasItem(
83                 allOf(hasProperty("id", is(2)), hasProperty("descuento", is(20)) ) )));
84     }
```

Comprueba el contenido de todos los objetos utilizando matchers

5-Prueba unitaria de un controlador web

Para invocar un POST basta con indicar
Los valores de los parámetros del form

```
86      * Situacion (2): Post indicando edad 19, devolvera una fila (de acuerdo con el mock)
94     @Test
95     public void testPostRequest() throws Exception {
96         //en el post debe especificar el parametro del form (campos DescuentoFormDTO)
97         ResultActions res=mvc.perform(post("/descuentos").param("edad", "19"))
98                 .andExpect(status().isOk());
99         res.andExpect(model().attribute("command", hasProperty("edad", is(19))));
100        res.andExpect(model().attribute("descuentos", hasSize(1)));
101        res.andExpect(model().attribute("descuentos", hasItem(
102                allOf(hasProperty("id", is(2)), hasProperty("descuento", is(20)) ) )));
103        //Las comparaciones se pueden realizar tambien usando los objetos del Model
104        @SuppressWarnings("unchecked")
105        List<DescuentoDisplayDTO> dto=(List<DescuentoDisplayDTO>) res.andReturn().getModelAndView().getModel().get("descuentos");
106        assertEquals(1,dto.size());
107        assertEquals(2,dto.get(0).getId().intValue());
108        assertEquals(20,dto.get(0).getDescuento().intValue());
109        //Con getResponse se obtiene el contenido de la respuesta, donde se puede
110        assertThat(res.andReturn().getResponse().getContentAsString(),
111                containsString("<title>Descuentos de clientes</title>"));
112    }
```

Comprueba el contenido de todos los objetos utilizando matchers

En vez de usar matchers, se pueden obtener todos los objetos realizar oos assert respecto de estos

Para comprobar la presencia de textos en la página
Se puede utilizar el html devuelto en la respuesta

6-Prueba de integración con el UI

Selenium

```
1  package giis.demo.descuento.it;
2⊕ import org.junit.*;
18
20⊕ * Pruebas de la interaccion del usuario con la aplicacion web    ejemplo de descu
43 @SpringBootTest(classes= {DescuentoApplication.class},
44     webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
45 @TestPropertySource(locations="classpath:application-test.properties")
46 @RunWith(SpringRunner.class)
47 public class TestDescuentoSelenium {
48     //datasource para acceso a la base de datos mediante sql con JdbcTemplate
49     @Autowired private javax.sql.DataSource datasource;
50     //Como se especifico WebEnvironment.RANDOM_PORT, esta variable identifica el pu
51     @LocalServerPort int port;
52     //Declara el driver de Selenium usado por las pruebas
53     WebDriver driver;
54
55     @Before
56     public void setUp() {
57         loadCleanDatabase();
58         loadMainPage("chrome");
59     }
60     @After
61     public void tearDown() {
62         driver.quit(); //cierra e
63     }
64     /**
65      * Inicializa el WebDriver para el navegador indicado y
66      */
67     private void loadMainPage(String browser) {
68         //Crea una instancia del driver que abrira el naveg
69         driver=SeleniumUtil.getNewDriver(browser);
70         //se dirige a la pagina principal
71         driver.get("http://localhost:"+port+"/");
72         //selecciona el link para ir a la pagina que se va a probar
73         driver.findElement(By.linkText("Ejecutar descuentos de clientes")).click();
74     }
```

Establece el entorno web para que despliegue el servidor Tomcat (en un puerto aleatorio para evitar interferencias)

Cada método se ejecutará con una BD limpia como antes y después se abre el navegador con la aplicación a probar.

Necesaria la limpieza aquí (cerrar navegador y finaliza sesión del driver). Si se quiere cerrar solo el navegador manteniendo la sesión usar close().

La inicialización consiste siempre en obtener instancia del driver (inicia la sesión) y navegar a la pagina (en este caso un menú que se selecciona). La obtención del driver está en un método reutilizable (ver código)

6-Prueba de integración con el UI

Selenium

```
76⊕  * Datos de prueba que se cargaran en e
80⊕ public void loadCleanDatabase() {
95
97⊕  * Escenario de prueba de la pantalla:
02⊖ @Test
03  public void testDescuentoScenario() {
04      doStep(true,"","Id,% Descuento
05              +"1,15\n"
06              +"2,20\n"
07              +"5,20\n"
08              +"6,10\n"
09              +"7,30\n");
10      doStep(false, "40", "Id,% Descuento\n"
11              +"5,20\n"
12              +"6,10\n");
13      doStep(false, "39", "Id,% Descuent
14              +"5,20\n"
15              +"6,10\n"
16              +"7,30\n");
17      doStep(false,"","Id,% Descuento\n"
18              +"1,15\n"
19              +"2,20\n"
20              +"5,20\n"
21              +"6,10\n"
22              +"7,30\n");
23  }
```

Como los pasos son similares, para evitar repetición de código
Se implementa un método común para todos los pasos
(como si fuera una prueba parametrizada

En cada paso se establece valor para la edad:
txtEdad=driver.findElement(By.id("txtEdad"));
se pulsa el botón y se comparan los resultados.
Como en el momento de cargar la pagina pueden no estar
Todos los elementos disponibles, usa WebDriverWait

```
128⊖ private void doStep(boolean initialStep, String edad, String expected) {
129     WebElement txtEdad;
130     txtEdad = (new WebDriverWait(driver, 5))
131         .until(ExpectedConditions.presenceOfElementLocated(By.id("txtEdad")));
132     if (initialStep) {
133         assertEquals("",txtEdad.getText()); //asegura que no hay texto
134     } else { //pone la edad y envia el formulario
135         txtEdad.clear(); //si no se limpia antes, sendKeys concatenara con el t
136         txtEdad.sendKeys(edad);
137         driver.findElement(By.id("btnEdad")).click();
138     }
139     //ilustra como guardar la imagen del navegador en este momento (el nombre i
140     SeleniumUtil.takeScreenshot(driver, initialStep+"-"+edad);
141     //busca la tabla en el navegador, obtiene el texto de las celdas y la compa
142     WebElement tab=driver.findElement(By.id("tabDescuentos"));
143     String[][] arrays=SeleniumUtil.getTableContent(tab);
144     assertEquals(expected, Util.arraysToCsv(arrays));
145 }
```

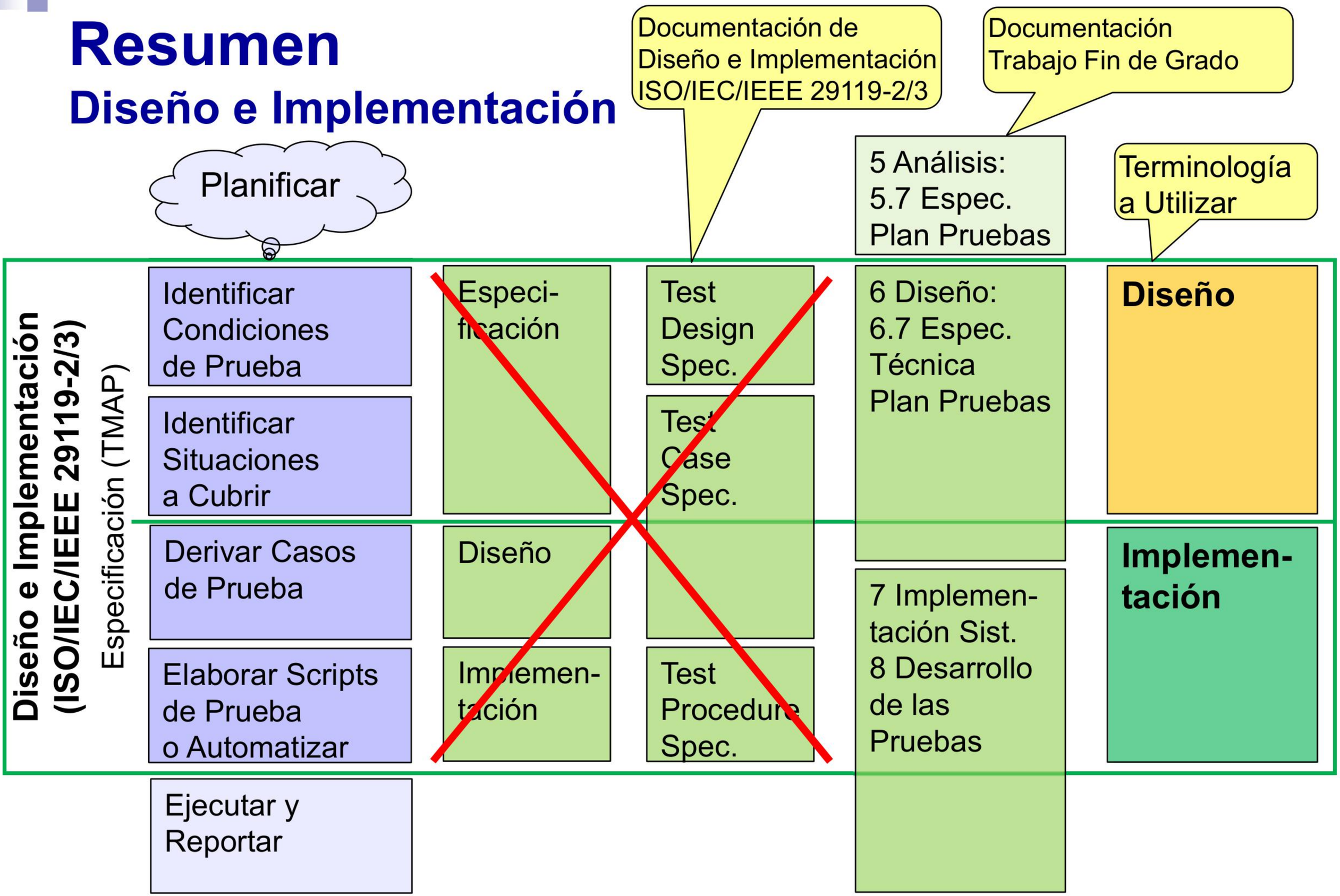
Ilustra uso de otros métodos de utilidad:

- Tomar una imagen de la pantalla
- Obtener todo el contenido de una tabla html (ver código)

Ojo: rendimiento al recorrer tablas flojo,
En getTableContents mejor usar jsoup
para hacer parse directamente del html

Resumen

Diseño e Implementación



Resumen

Posible transformación siguiendo alguna **estrategia de combinación**. Puede haberse pensado de esta Forma directamente

Trazabilidad: qué situaciones (hijos) son ejercitadas por cada caso de prueba

Problema 3
- Tipo de cliente
  - Nuevo
  - Existente
- Cupón dto.
  - Si
  - No
- Tarjeta fideliz.
  - Si
  - No

Problema 1
- Importe del principal
  - Entre 10K y 50K
    - 10.000,00
    - 49.999,99

Problema 3
- Nuevo Cliente
  - Sin cupón ni tarjeta
  - Con cupón
  - Con tarjeta (inv.)
- Cliente existente
  - Sin cupón ni tarjeta
  - Con cupón
  - Con tarjeta

Situaciones a Cubrir (Test Coverage Items)

Casos de Prueba (Test Cases)

Diseño:
Definir lo que se va a probar:
Estructura jerárquica con las diferentes **situaciones a cubrir**

Definir lo que hay que probar: **Condiciones de Prueba** (Test Conditions) de entrada y/o salida. Se derivan sus **Clases de Equivalencia** y **Valores Límite**, si aplicable

Implementación:
De qué forma se organizará la prueba:
Derivar **casos de prueba**
(los menos posibles)
para cubrir todas las situaciones

Resumen

Describe en términos lógicos las circunstancias en las que se probará El comportamiento del sistema, incluyendo las situaciones a cubrir [TMAP]

Descripción detallada de casos de prueba para poder ejecutarlos de forma eficiente y simple: Documento, Hoja Excel, Registro en una herramienta...

Situaciones a Cubrir (Test Coverage Items)

Trazabilidad

Casos de Prueba (Test Cases)

Caso de prueba Lógico

- Entradas (Combinaciones tipo cliente y cupones/fidelizac
  - Nuevo Cliente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta fidelización (inválida)
  - Cliente existente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta de fidelización
- Salidas
  - Acumulación de descuentos
    - Se acumulan
    - No se acumulan
  - Descuento aplicado (%)
    - 15
    - 10
    - 20
    - 0

Casos de prueba
Caso 1
Caso 2
Caso 3
Caso 4
Caso 5
Caso 6
Caso 7

Caso de prueba Físico

- Entradas
  - Nuevo Cliente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta fidelización (inválida)
  - Cliente existente
    - Sin cupón descuento ni tarjeta fideliz.
    - Con cupón descuento
    - Con tarjeta de fidelización
- Salidas
  - Acumulación de descuentos
    - Se acumulan
    - No se acumulan
  - Descuento aplicado (%)
    - 15

Ahora:
1 caso de prueba

(Base de Datos)

ID Cli.	Nuevo	Cupon	Tarjeta
1	S	N	N
2	S	S	N
3	S	N	S
4	N	N	N
5	N	S	N
6	N	N	S
7	N	S	S

(Informe)

ID Cli.	% Dto.
1	15
2	20
5	20
6	10
7	30

Script para Ejecución Manual

Script para Ejecución Automática

JUnit

Elaboración concreta del caso lógico con todos los valores de las entradas, salidas y configuración del entorno [TMAP]

Implementación en un programa o configuración de casos de prueba para ejecución utilizando una herramienta

Resumen

- Técnicas básicas e intuitivas. Proceso:
  - Determinar las **condiciones de prueba** (*test conditions*) para entradas. Completar con salidas.
  - Determinar y aplicar técnicas para determinar las **situaciones a cubrir** (*test coverage ítems*).
  - Decidir si algunas situaciones a cubrir se han de combinar. Compromiso coste/beneficio.
  - Partiendo de las situaciones a cubrir, derivar los **casos de prueba** (lo más mecánico).
  - Al ejecutar los casos de prueba, no olvidar comparar lo que cambia y lo que no debería cambiar (incluyendo actualizaciones de base de datos)
- Notas:
  - Cuando hay base de datos, algunas situaciones a probar se representan en filas en la base de datos
  - Cuando hay interfaz de usuario puede haber casos de prueba compuestos por varios pasos (no abusar)
  - En general: Nº situaciones > Nº casos > Nº bases de datos