CFGS Desenvolupament d'Aplicacions Multiplataforma (DAM)

CFGS Desenvolupament d'Aplicacions Web (DAW)

Mòdul 5 – Entorns de Desenvolupament.
UF2 – Optimització de programari

Exemples de disseny de casos de prova

Índex

Índex de contingut

EXEMPLE 1: CUA DE LLETRES

En aquest exemple es contemplen dos casos:

- Els paràmetres vàlids formen part d'un rang (lletres) dins d'un conjunt finit (caràcters ASCII).
- El paràmetre és una llista de valors on importa especialment la seva longitud.

Hem implementat una cua de lletres. Representa una cua (FIFO) d'elements. Cada element és una lletra. Té els següents mètodes:

Constructor: sense paràmetres; crea una cua buida (sense cap element).

void posa(**char** c): afegeix la lletra *c* a la cua (l'afegeix al final, com a darrer element). Si *c* no conté una lletra acaba amb una excepció del tipus NoLletraException.

char consulta(): retorna la lletra de la cua que fa més temps que hi és (és a dir, la primera de la cua). Si la cua és buida, acaba amb una excepció del tipus CuaLletresBuidaException.

char treu(): treu la primera lletra de la cua i la retorna com a resultat. Si la cua és buïda, acaba amb una excepció del tipus CuaLletresBuidaException.

boolean esBuida(): retorna cert si la cua és buïda (és a dir, no té cap element) i fals en cas contrari.

La implementació és la següent:

```
package ioc.eac2.cua;

import java.util.NoSuchElementException;
import java.util.Vector;


/**
 * @author IOC
 *
 */
public class CuaLletres {

	private Vector<Character> v= new Vector<Character> ();

	/**
	 * Afegeix la lletra c al final de la cua.
	 * El caracter <i>c</i> ha de ser una lletra, majuscula o minuscula.
	 * @param c lletra que s'afegeix.
	 * @throws NoLletraException si c no conte una lletra
	 */
	public void posa(char c) throws NoLletraException {

		if(Character.isLetter(c))
			v.addElement(c);
		else
			throw		new NoLletraException();
	}
```

```
	/**
	 * Retorna la lletra que ocupa la primera posicio de la cua.
	 * La cua no ha de ser buida.
	 * @return la lletra que ocupa la primera posicio de la cua.
	 * @throws CuaLletresBuidaException si la cua es buida
	 */
	public char consulta() throws CuaLletresBuidaException {
		try{

			return v.firstElement();

		} catch (NoSuchElementException e)  {

				throw new CuaLletresBuidaException () ;

			}
	}

	/**
	 * Elimina la primera lletra de la cua.
	 * La cua no ha de ser buida.
	 * @return la lletra eliminada
	 * @throws CuaLletresBuidaException si la cua es buida
	 */
	public char treu() throws CuaLletresBuidaException {
		try {
			return v.remove(0);
		} catch(ArrayIndexOutOfBoundsException e) {

				throw new CuaLletresBuidaException();

			}
	}
	/**
	 * Retorna cert si la cua es buida.  Fals en cas contrari.
	 * @return Retorna cert si la cua es buida.  Fals en cas contrari.
	 */
	public boolean esBuida()	{

		return v.isEmpty();

	}

}
```

```
package ioc.eac2.cua;

/**
 * @author ioc
 * Classe que representa l'excepció produïda quan un valor havia de ser una
   lletra i no ho és
 *
 */
@SuppressWarnings("serial")
public class NoLletraException extends Exception {

}
```

```
package ioc.eac2.cua;

/**
 * @author ioc
 * Classe que representa l'excepció produïda per trobar-se la cua de lletres
buida
 *
 */
@SuppressWarnings("serial")
public class CuaLletresBuidaException extends Exception {



}
```

Cal dissenyar els casos de prova per comprovar el funcionament correcte de la classe. Cal fer el següent:

a) Disseny de les classes d'equivalència. Cal seguir els passos que s'indiquen al punt del material 1.3.2 (Disseny de les proves. Tipus de proves), apartat “Classes d'equivalència”(és a dir, els passos són: “Identificar les condicions, restriccions o continguts de les entrades i les sortides” i ”Identificar, a partir de les condicions, les classes d’equivalència de les entrades i les sortides”). Per cada mètode caldrà trobar les seves classes d'equivalència.

SOLUCIÓ PROPOSADA:

`public void posa(char c) throws NoLletraException`

Condicions:

- D'entrada: *c* ha de ser una lletra
- De sortida: si *c* no és cap lletra, llença una excepció

Classes d'equivalència (cal tenir present la codificació dels caràcters):

1. Entrada vàlida 1: *c* és una lletra majúscula ('A' ≤ *c* ≤ 'Z')
2. Entrada vàlida 2: *c* és una lletra minúscula ('a' ≤ *c* ≤ 'z')
3. Entrada invàlida 1: *c* és un caràcter menor que les lletres (*c* < 'A')
4. Entrada invàlida 2: *c* és un caràcter comprés entre les majúscules i les minúscules ('Z' < *c* < 'a')
5. Entrada invàlida 3: *c* és un caràcter més gran que les lletres ('z' < *c*)

Les classes d'equivalència que es generarien a partir de les condicions de sortida ja van incloses a les classes anteriors (*cap sortida* a les entrades vàlides i *llençar una excepció* a les entrades invàlides).

`public char consulta() throws CuaLletresBuidaException`

Condicions:

- D'entrada: la cua no ha de ser buida
- De sortida: si la cua és buida, es genera una excepció; sinó, es retorna la primera lletra

Classes d'equivalència:

1. Entrada vàlida: cua amb algun element
2. Entrada invàlida: cua sense cap element

Les classes d'equivalència que es generarien a partir de les condicions de sortida ja van incloses a les classes anteriors (*la primera lletra* a *entrada vàlida* i *llençar una excepció* a *entrada invàlida*).

```
public char treu() throws CuaLletresBuidaException
```

Condicions:

- D'entrada: la cua no ha de ser buida.
- De sortida: si la cua és buida, es genera una excepció

Classes d'equivalència:

1. Entrada vàlida: cua amb algun element
2. Entrada invàlida: cua sense cap element

Les classes d'equivalència que es generarien a partir de les condicions de sortida ja van incloses a les classes anteriors (*la primera lletra* a *entrada vàlida* i *llençar una excepció* a *entrada invàlida*).

```
public boolean esBuida()
```

Condicions:

- D'entrada: no hi ha cap condició especial → no tindrem cap classe d'equivalència que depengui de l'entrada.
- De sortida: la sortida només pot valer *true* o *false*.

Classes d'equivalència:

1. Sortida vàlida 1: la sortida és *true*
2. Sortida vàlida 2: la sortida és *false*

b) Dissenyeu els casos de prova a partir de les classes seleccionades anteriorment. Tingueu en compte que:

- Com a mínim ha d'haver un representant de cada classe d'equivalència
- Han de cobrir els valors límit i els errors típics
- Han de recórrer almenys una vegada cada camí independent (això últim està relacionat amb els apartats “Cobertura del flux de control” i “Complexitat ciclomàtica”).

SOLUCIÓ PROPOSADA:

Tot i que Java treballa amb Unicode, només considerarem els caràcters que es corresponen amb la codificació ASCII, ja que són els que es poden entrar des del teclat estàndard.

```
public void posa(char c) throws NoLletraException
```

Classes d'equivalència:

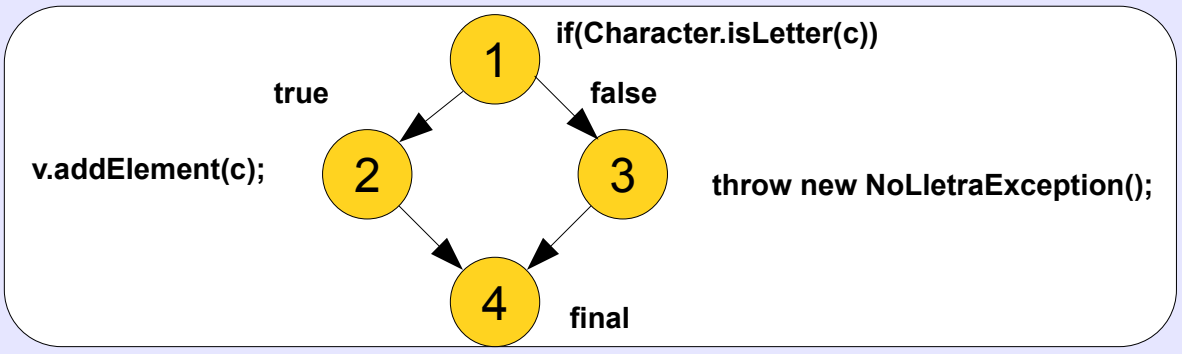
Entrada vàlida 1: *c* és una lletra majúscula ('A' ≤ *c* ≤ 'Z').
Casos de prova:
- Valors límits: *c* = 'A' i *c* = 'Z'
- Valor intermedi: *c* = 'M'

Entrada vàlida 2: *c* és una lletra minúscula ('a' ≤ *c* ≤ 'z')
Casos de prova:
- Valors límits: *c* = 'a' i *c* = 'z'
- Valor intermedi: *c* = 'm'

Entrada invàlida 1: *c* és un caràcter menor que les lletres (*c* < 'A')
Casos de prova:
- Valors límits: *c* = ' ' i *c* = '@'
- Valor intermedi: *c* = '1'

Entrada invàlida 2: *c* és un caràcter comprés entre les majúscules i les minúscules ('Z' < *c* < 'a')
Casos de prova:
- Valors límits: *c* = '[' i *c* = '` '
- Valor intermedi: *c* = '^'

Entrada invàlida 3: *c* és un caràcter més gran que les lletres ('z' < *c*)
Casos de prova:
- Valors límits: *c* = '{' i *c* = '~'
- Valor intermedi: *c* = '}'



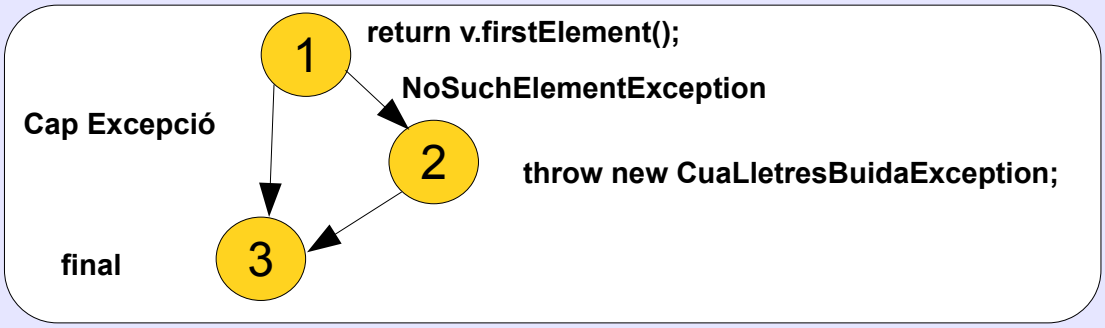
Càlcul de la complexitat ciclomàtica: nombre de branques – nombre de nodes + 2 = 4 – 4 + 2 = 2
Els camins serien: 1-2-4 i 1-3-4.

Amb els casos anteriors estan coberts:
1-2-4 amb 'M'
1-3-4 amb '@'

`public char consulta() throws CuaLletresBuidaException`

Classes d'equivalència:

1. Entrada vàlida: cua amb algun element
   Casos de prova:
   - Primer valor: cua amb un sol element.
   - Últim valor: no hi ha límit
   - Valor intermedi: la cua amb 3 elements (podria ser qualsevol altre nombre d'elements)
2. Entrada invàlida: cua sense cap element
   Cas de prova:
   - La cua buida



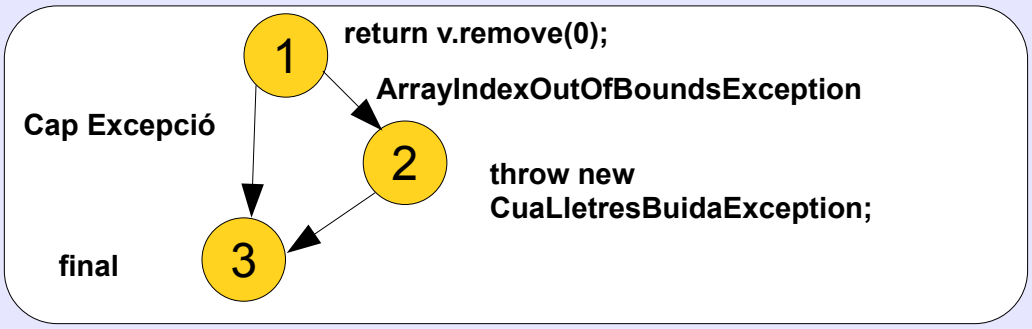
Càlcul de la complexitat ciclomàtica: nombre de branques – nombre de nodes + 2 = 3 – 3 + 2 = 2
Els camins serien: 1-2-3 i 1-3.

Amb els casos anteriors estan coberts:
1-2-3: amb cua amb un sol element
1-3: amb cua buida

`public char treu() throws CuaLletresBuidaException`

Classes d'equivalència:
1. Entrada vàlida: cua amb algun element
   Casos de prova:
   - Primer valor: cua amb un sol element.
   - Últim valor: no hi ha límit
   - Valor intermedi: la cua amb 3 elements (podria ser qualsevol altre nombre d'elements)

2. Entrada invàlida: cua sense cap element
   Cas de prova:
   - La cua buida



Càlcul de la complexitat ciclomàtica: nombre de branques – nombre de nodes + 2 = 3 – 3 + 2 = 2
Els camins serien: 1-2-3 i 1-3.

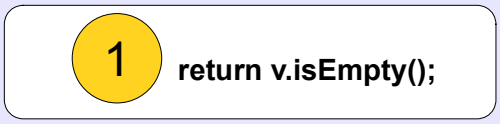
Amb els casos anteriors estan coberts:
1-2-3 amb una cua d'un sol element
1-3 amb una cua buida

```
public boolean esBuida()
```

Classes d'equivalència:

1. Sortida vàlida 1: la sortida és *true*
   Cas de prova:
   - La cua buida

2. Sortida vàlida 2: la sortida és *false*
   Casos de prova:
   - Valors extrems: cua amb un sol element (no hi ha límit superior)
   - Valor intermedi: *la cua amb 3 elements (podria ser qualsevol altre nombre d'elements)*



Càlcul de la complexitat ciclomàtica: nombre de branques – nombre de nodes + 2 = 0 – 1 + 2 = 1
Només hi ha un camí possible. Amb qualsevol dels anteriors casos està cobert.

Observació: en la solució hem seguit estrictament les indicacions del material. Si utilitzem una llibreria que ja ha estat provada altres vegades, podriem haver simplificat les proves (p.e. el mètode *posa* crida a *Character.isLetter.* Si ja tenim comprovat que fa el que esperem, podem senzillament agafar una entrada que sigui lletra i una que no ho sigui).

EXEMPLE 2: INCREMENTAR NOTA

En aquest exemple es contemplen dos casos:
- Els paràmetres vàlids formen part d'un rang (0-10) dins d'un conjunt teòricament infinit (enters).
- Els paràmetres vàlids és una llista de valors no consecutius que pertanyen a un tipus amb infinits valors.

Tenim una classe que conté dos mètodes que implementen una utilitat que incrementa una nota en un grau o una unitat (en funció de si la nota és entera o qualitativa, respectivament). També tenim una classe que implementa un tipus d'excepció que es llença quan el valor de la qualificació no és correcte.

Les classes són aquestes:

```
package exemplesproves;

public class GestionaQualificacions {
/**
 * Retorna el resultat d'incrementar en una unitat una nota entera compresa
entre el 0 i el 10, tots dos inclosos.
 * Si el parametre es incorrecte, llenca una excepcio;
 * si el parametre val 10 retorna el mateix valor.
 * @param nota nota a incrementar
 * @return resultat d'incrementar la nota en una unitat
 * @throws NotaIncorrectaException si la nota no esta entre 0 i 10 (inclosos)
 */
    public static int incrementa(int nota) throws NotaIncorrectaException{
        if(nota>=0 && nota < 10){
            return nota+1;
        }else if (nota==10){
            return 10;  // no es pot incrementar
        }else{
            throw new NotaIncorrectaException();
        }
    }
/**
 * Retorna el resultat d'incrementar en un grau una nota qualitativa
 * Si el parametre es incorrecte, llenca una excepcio;
 * si el parametre te el valor maxim, el mateix valor.
 * @param nota nota a incrementar; pot ser "Malament", "Be", "Molt be".
 * @return resultat d'incrementar la nota en un grau
 * @throws NotaIncorrectaException si la nota no val "Malament", "Be" o
   "Molt be"
 */
    public static String incrementa(String nota) throws NotaIncorrectaException
    {
        if(nota.equals("Malament")){
            return "Be";
        }else if (nota.equals("Be")){
            return "Molt be";
        }else if(nota.equals("Molt be")){
            return "Molt be"; // no es pot incrementar
        }else{
            throw new NotaIncorrectaException();
        }
    }
}
```

```
package exemplesproves;
/**
 * Excepcio que es llenca quan el valor d'una nota no es correcte
 * @author professor
 */
public class NotaIncorrectaException extends Exception {

}
```

Volem dissenyar els casos de prova per comprovar el funcionament correcte de la classe *GestionaQualificacions*. Cal fer el següent:

a) Disseny de les classes d'equivalència. Cal seguir els passos que s'indiquen al punt del material 1.3.2 (Disseny de les proves. Tipus de proves), apartat "Classes d'equivalència"(és a dir, els passos són: "Identificar les condicions, restriccions o continguts de les entrades i les sortides" i "Identificar, a partir de les condicions, les classes d'equivalència de les entrades i les sortides"). Per cada mètode caldrà trobar les seves classes d'equivalència.

SOLUCIÓ PROPOSADA:

```
public static int incrementa(int nota) throws NotaIncorrectaException
```

Condicions:

- D'entrada: *nota* ha d'estar compresa entre 0 i 10, tots dos inclosos.
- De sortida: si *nota* no està compresa entre 0 i 10, tots dos inclosos, llença una excepció

Classes d'equivalència (cal tenir present la codificació dels caràcters):

1. Entrada vàlida: *nota* és un enter comprés entre el 0 i el 10 ($0 \leq nota \leq 10$)
2. Entrada invàlida 1: *nota* és un enter menor que 0 ($nota < 0$)
3. Entrada invàlida 2: *nota* és un enter major que 10 ($nota > 10$)

Les classes d'equivalència que es generarien a partir de les condicions de sortida ja van incloses a les classes anteriors (la nota incrementada en 1 unitat -o la mateixa, si el paràmetre val 10- a les entrades vàlides i *llençar una excepció* a les entrades invàlides).

```
public static String incrementa(String nota) throws NotaIncorrectaException
```

Condicions:

- D'entrada: *nota* ha de valer "Malament", "Be" o "Molt be".
- De sortida: si *nota* no és cap dels valors anteriors, llença una excepció

Classes d'equivalència:

1. Entrada vàlida 1: *nota* val "Malament"
2. Entrada vàlida 2: *nota* val "Be"
3. Entrada vàlida 3: *nota* val "Molt be"
4. Entrada invàlida: *nota* no val cap dels valors anteriors.

Les classes d'equivalència que es generarien a partir de les condicions de sortida ja van incloses a les classes anteriors (la nota augmentada un grau -o la mateixa, si el paràmetre val "Molt be"- a les entrades vàlides i *llençar una excepció* a les entrades invàlides).

b) Dissenyeu els casos de prova a partir de les classes seleccionades anteriorment. Tingueu en compte que:

- Com a mínim ha d'haver un representant de cada classe d'equivalència
- Han de cobrir els valors límit i els errors típics
- Han de recórrer almenys una vegada cada camí independent (això últim està relacionat amb els apartats "Cobertura del flux de control" i "Complexitat ciclomàtica").

SOLUCIÓ PROPOSADA:

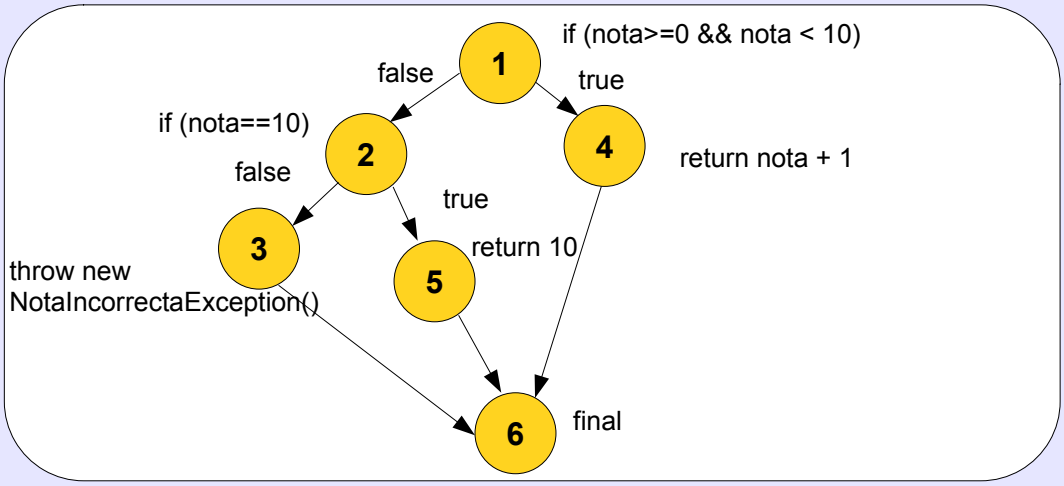
```
public static int incrementa(int nota) throws NotaIncorrectaException
```

Classes d'equivalència:

Entrada vàlida: *nota* és un enter comprés entre 0 i 10, tots dos inclosos (0 ≤ *nota* ≤ 10).
Casos de prova:
- Valors límits: *nota* = *0* i *nota* = 10
- Valor intermedi: *nota* = 5

Entrada invàlida 1: *nota* és un enter menor que 0
Casos de prova:
- Valor límit: *nota* = -1
- Valor intermedi: com no és trivial trobar el límit inferior (ni, per tant, el valor intermedi), podem agafar qualsevol enter menor que zero; per exemple: *nota* = -200

Entrada invàlida 2: *nota* és un enter major que 10
Casos de prova:
- Valor límit: *nota* = 11
- Valor intermedi: com no és trivial trobar el límit superior (ni, per tant, el valor intermedi), podem agafar qualsevol enter major que deu; per exemple: *nota* = 300



Càlcul de la complexitat ciclomàtica: nombre de branques – nombre de nodes + 2 = 7 – 6 + 2 = 3
Els camins serien: 1-2-3-6, 1-2-5-6 i 1-4-6.

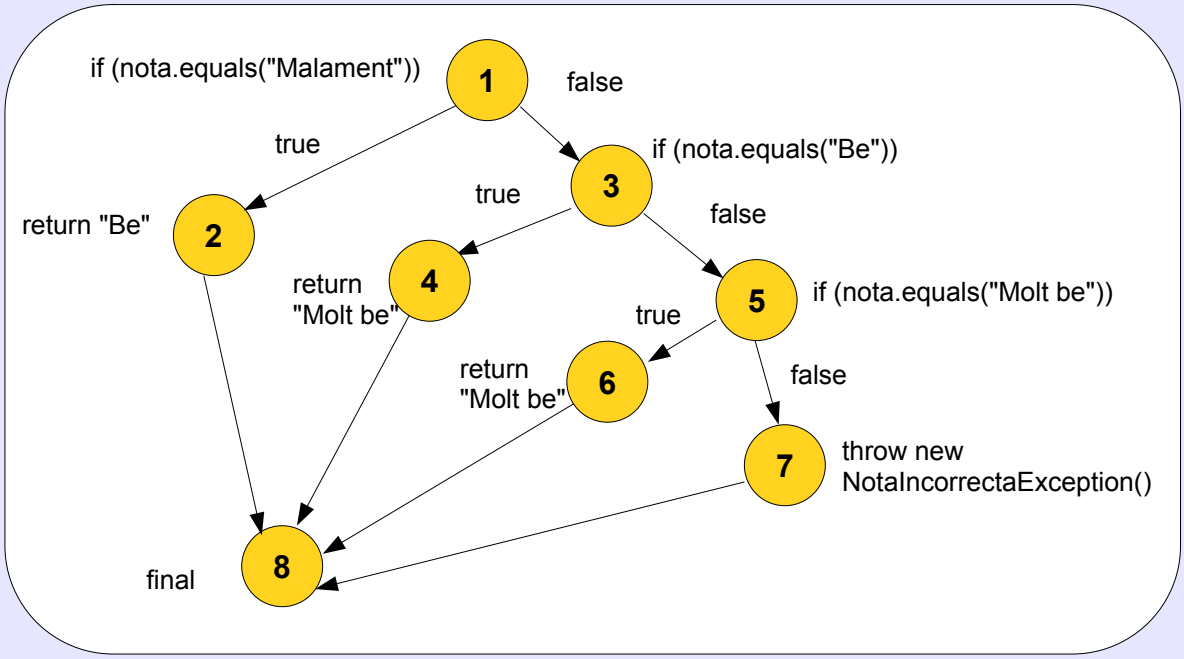
Amb els casos anteriors estan coberts:
1-2-3-6 amb *nota* = -200 o *nota* = 300 (en aquest cas no cal posar més d'un)
1-2-5-6 amb *nota* = 10
1-4-6 amb *nota* = 0 o *nota* = 10 o *nota* = 5 (en aquest cas no cal posar més d'un)

```
public static String incrementa(String nota) throws NotaIncorrectaException
```

Classes d'equivalència:

Classes d'equivalència:
1. Entrada vàlida 1: *nota* val “Malament”. Cas de prova *nota*=”Malament”.
2. Entrada vàlida 2: *nota* val “Be”. Cas de prova *nota*=”Be”.
3. Entrada vàlida 3: *nota* val “Molt be”. Cas de prova *nota*=”Molt be”.
4. Entrada invàlida: *nota* no val cap dels valors anteriors.

Casos de prova:
- Valors límit (suposant que la codificació dels caràcters és ASCII):
  - “Malamen~” i “Malament “
  - “B~” i “Be “
  - “Molt b~” i “Molt be “
- Valor intermedi: com no és trivial trobar-lo podem agafar valors qualsevol que estiguin entre els diferents valors correctes; per exemple: “Guai” (entre “Be” i “Malament”), “Meca” entre “Malament” i “Molt be”).



Càlcul de la complexitat ciclomàtica: nombre de branques – nombre de nodes + 2 = 10 – 8 + 2 = 4
Els camins serien: 1-3-5-7-8, 1-3-5-6-8, 1-3-4-8, 1-2-8.

Amb els casos anteriors estan coberts:
1-3-5-7-8 amb *nota* = “Guai” (per exemple)
1-3-5-6-8 amb *nota* = “Molt be”
1-3-4-8 amb *nota* = “Be”
1-2-8 amb amb *nota* = “Malament”

EXEMPLE 3: COMPARAR DAUS

En aquest exemple verificarem un mètode que té més d'un paràmetre.

Tenim una classe que conté un mètode que rep com a paràmetre dos valors que representen dues tirades de dau i que retorna cert si tots dos valors són iguals, fals en cas contrari; si algun dels paràmetres o tots dos està fora del rang lògic (el dau té 6 cares numerades de l'1 al 6) llença una excepció *DausForaRangException*. Volem verificar aquest mètode.

La classe és aquesta:

```
package exemplesproves;

public class GestioDaus {

      /**
       * Indica si dues tirades de dau donen el mateix resultat
       * @param primer primer valor a comparar; representa una
       * tirada de dau i, per tant, ha d'estar entre 1 i 6
       * @param segon segon valor a comparar; representa una tirada de dau i,
       * per tant, ha d'estar entre 1 i 6
       * @return cert si tots dos valors son iguals; fals en cas contrari
       * @throws DausForaRangException si un dels dos valors o tots dos son
       * fora dels valors correctes
       */

    public static boolean verificaEmpat(int primer, int segon)
                                                    throws DausForaRangException{
        if(primer<1 || primer>6 || segon <1 || segon>6){
            throw new DausForaRangException();
        }else if(primer==segon){
            return true;
        }else{
            return false;
        }
    }


}
```

```
package exemplesproves;

/**
 * Excepcio que es llenca quan el valor d'una nota no es correcte
 * @author professor
 */

public class DausForaRangException extends Exception {

}
```

Volem dissenyar els casos de prova per comprovar el funcionament correcte del mètode *verificaEmpat*. Cal fer el següent:

a) Disseny de les classes d'equivalència. Cal seguir els passos que s'indiquen al punt del material 1.3.2 (Disseny de les proves. Tipus de proves), apartat “Classes d'equivalència”(és a dir, els passos són: “Identificar les condicions, restriccions o continguts de les entrades i les sortides” i ”Identificar, a partir de les condicions, les classes d’equivalència de les entrades i les sortides”). En aquest cas, en tenir dos paràmetres el mètode, caldrà trobar les classes d'equivalència per a tots dos paràmetres.

SOLUCIÓ PROPOSADA:

```
public static boolean  verificaEmpat(int primer, int segon)
                                                  throws  DausForaRangException
```

Condicions:

- D'entrada: *primer* i *segon* han d'estar tots dos compresos entre 1 i 6, tots dos inclosos.
- De sortida: si algun (o tots dos) dels paràmetres no estan compresos entre 1 i 6, tots dos inclosos, llença una excepció; sinó, si tots dos paràmetres són iguals, retorna cert i fals en cas contrari.

Classes d'equivalència:

1. Entrada vàlida 1: *primer* i *segon* són tots dos enters compresos entre l'1 i el 6 ($1 \leq primer \leq 6$ i $1 \leq segon \leq 6$).
2. Entrada invàlida 1: *primer* és un enter menor que 1 ($primer < 1$)
3. Entrada invàlida 2: *primer* és un enter major que 6 ($primer > 6$)
4. Entrada invàlida 3: *segon* és un enter menor que 1 ($primer < 1$)
5. Entrada invàlida 4: *segon* és un enter major que 6 ($primer > 6$)
6. Sortida vàlida 1: retorna cert
7. Sortida vàlida 2: retorna fals

Les classes d'equivalència que es generarien a partir de la resta de condicions de sortida ja van incloses a les classes anteriors (les classes vàlides fan que es retorni un booleà, les invàlides generen una excepció). Aquestes dues classes de sortida s’han considerat perquè els seus membres són tractats diferent pel programa (provoquen l'execució de branques dels *if* diferents).

b) Dissenyeu els casos de prova a partir de les classes seleccionades anteriorment. Tingueu en compte que:

- Com a mínim ha d'haver un representant de cada classe d'equivalència
- Han de cobrir els valors límit i els errors típics
- Han de recórrer almenys una vegada cada camí independent (això últim està relacionat amb els
- apartats “Cobertura del flux de control” i “Complexitat ciclomàtica”).

SOLUCIÓ PROPOSADA:

```
public static boolean  verificaEmpat(int primer, int segon)
                                        throws  DausForaRangException
```

Classes d'equivalència:

Entrada valida 1: *primer* i *segon* són tots dos enters compresos entre l'1 i el 6 (*1 ≤ primer ≤ 6 i 1 ≤ segon ≤ 6*).

Casos de prova: cal combinar els valors límits i intermedi dels dos paràmetres. Aquests valors són:
- Valors limits: *primer = 1, primer = 6, segon = 1* i *segon = 6*
- Valors intermedis: *primer = 3* i *segon = 3.*

Combinant aquests valors (tots els que hem assenyalat per *primer* amb tots els que hem assenyalat per *segon*), surten els següents casos de prova:

- *primer = 1* i *segon = 1*
- *primer = 1* i *segon = 3*
- *primer = 1* i *segon = 6*
- *primer = 3* i *segon = 1*
- *primer = 3* i *segon = 3*
- *primer = 3* i *segon = 6*
- *primer = 6* i *segon = 1*
- *primer = 6* i *segon = 3*
- *primer = 6* i *segon = 6*

Entrada invalida 1: *primer* es un enter menor que 1 (*primer* < 1)
Casos de prova:
- Valor limit: *primer = 0*
- Valor intermedi: en aquest cas, qualsevol altre inferior a 1; p.e. -8

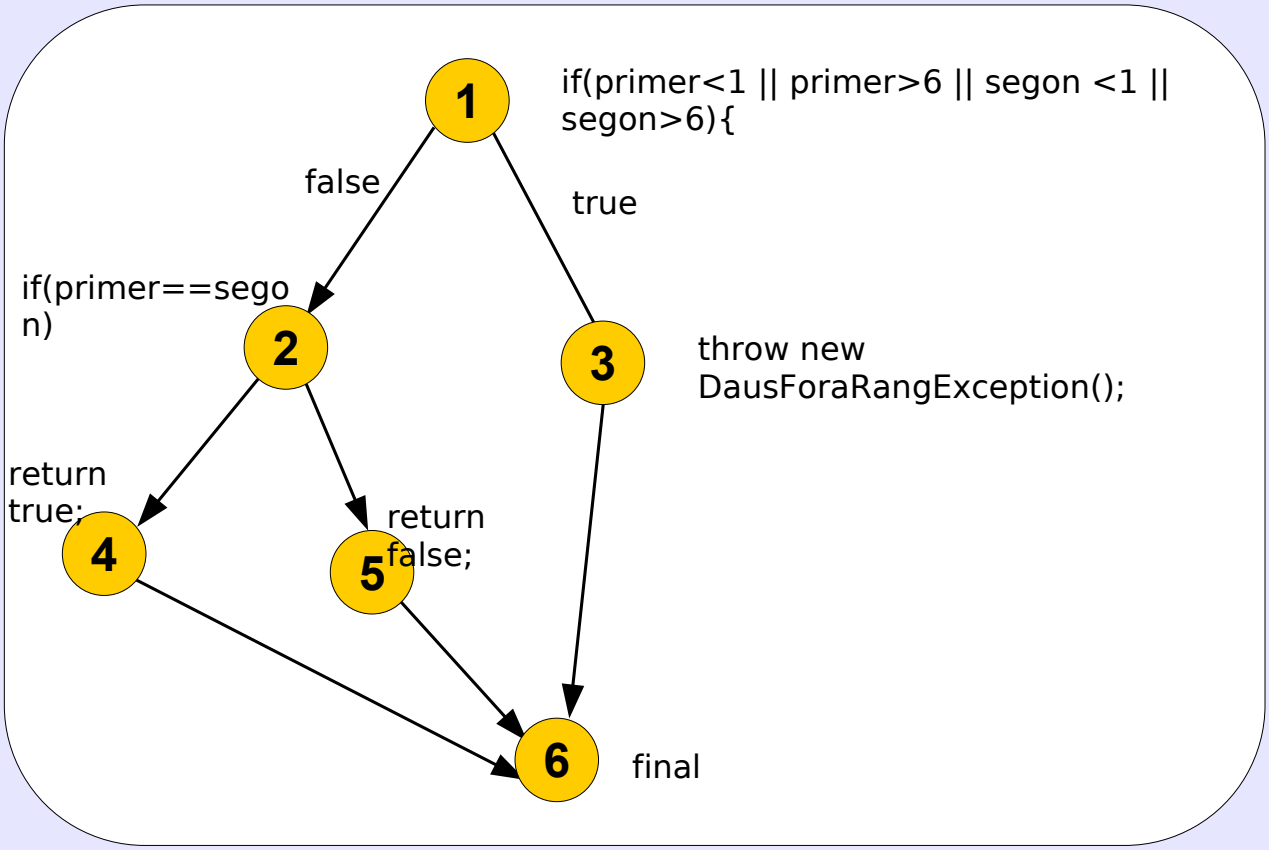
Entrada invalida 2: *primer* es un enter major que 6 (*primer* > 6)
Casos de prova:
- Valor limit: *primer = 7*
- Valor intermedi: en aquest cas, qualsevol altre superior a 6; p.e. 20

Entrada invalida 3: *segon* es un enter menor que 1 (*segon* < 1)
Casos de prova:
- Valor limit: *segon = 0*
- Valor intermedi: en aquest cas, qualsevol altre inferior a 1; p.e. -8

Entrada invalida 4: *segon* es un enter major que 6 (*segon* > 6)
Casos de prova:
- Valor limit: *segon = 7*
- Valor intermedi: en aquest cas, qualsevol altre superior a 6; p.e. 20

Sortida valida 1: retorna cert
Cas de prova: qualsevol on *primer* i *segon* siguin valids i iguals.
Per exemple: *primer* = 2 i *segon* = 2

Sortida valida 2: retorna fals
Cas de prova: qualsevol on *primer* i *segon* siguin valids i diferents.
Per exemple: *primer* = 3 i *segon* = 6



Càlcul de la complexitat ciclomàtica: nombre de branques - nombre de nodes + 2 = 7 - 6 + 2 = 3

Els camins serien: 1-2-4-6, 1-2-5-6 i 1-3-6.

Amb els casos anteriors estan coberts:

1-3-6 amb *primer* = 0
1-2-5-6 amb *primer* = 3 i *segon* = 6
1-2-4-6 amb *primer* = 2 i *segon* = 2