Curs 2019-2020

UD 4. Eines d'automatització

Automatització en Java. Apache ant.

Entorns de desenvolupament

IES Jaume II El Just
Tavernes de la Valldigna

Continguts

1. Introducció

En Java disposem de diverses eines que ens ajuden a automatitzar les tasques de construcció d'aplicacions i gestionar les seues dependències.

En aquesta part de la unitat veurem tres de les principals eines: Apache Ant, Apache Maven i Gradle. Veurem les principals característiques d'elles, i aprendrem com crear i gestionar el cicle de construcció de l'aplicació amb cadascuna.

Posteriorment, veurem com incloure llibreríes en els nostres projectes i gestionar-ne les dependències, de manera que l'usuari o altres desenvolupadors no s'hagen de preocupar de descarregar-les i instal·lar-les pel seu compte.

Aplicació d'exemple

Com a aplicació Java a utilitzar en l'exemple, anem a implementar l'aplicació de la calculadora que vam veure amb el make. Per a això, disposem de dos fitxers .java, organitzats de ls següent forma:

```
.
`-- com
    `-- ieseljust
        `-- edd
            `-- calc
                |-- Calculadora.java
                `-- Calcula.java
```

Com veiem, el codi font està organitzat en una estructura de carpetes. Açò ens serveix per organitzar millor el codi de les nostres aplicacions. Generalment, les carpetes de més alt nivell especifiquen l'organització, i internament van estructurant-se segons la funcionalitat. Com podeu veure, es tracta com si fos un domini d'Intenet, però de l'inrevés (com/ieseljust/edd -> edd.ieseljust.com).

Aquesta estructura, ens determinarà el *nom del paquet* al que pertànyen els programes. Anem a veure el codi de cadascun.

Classe com.ieseljust.edd.calc.Calculadora

Aquesta classe s'implementa al fitxer `com/ieseljust/edd/calc/Calculadora.java`:

```java
package com.ieseljust.edd.calc;

public class Calculadora {

    private float lastResult;
```

```
6       private String lastOp;
7
8       public float getLastResult(){
9           return this.lastResult;
10      }
11
12      public String getLastOp(){
13          return this.lastOp;
14      }
15
16      public float suma(float op1, float op2){
17          float result=op1+op2;
18          this.lastResult=result;
19          this.lastOp="Suma";
20
21          return result;
22      }
23
24      public float resta(float op1, float op2){
25          float result=op1-op2;
26          this.lastResult=result;
27          this.lastOp="Resta";
28
29          return result;
30      }
31
32      public float multiplica(float op1, float op2){
33          // Fem els càlculs
34          float result=op1*op2;
35
36          // Actualitzem els atributs de la classe
37          this.lastResult=result;
38          this.lastOp="Multiplica";
39
40          // I finalment retornem els resultats
41          return result;
42      }
43
44      public float divideix(float op1, float op2){
45          // Fem els càlculs
46          float result=op1/op2;
47
48          // Actualitzem els atributs de la classe
49          this.lastResult=result;
50          this.lastOp="Divideix";
51
52          // I finalment retornem els resultats
53          return result;
54      }
55
56  }
```

Anem a explicar-lo per parts:

- En primer lloc, s'indica el nom del paquet al què pertany el programa, de forma completa. Fixeu-se que ve definit per l'estructura de directoris on es troba:

```
1 package com.ieseljust.edd.calc;
```

- Després es defineix la classe Calculadora. Recordeu que en Java tot ha d'estar dins una classe, tant el main com les funcions (mètodes) que definim. L'estructura general d'aquesta classe `Calculadora` és la següent (mostrem només algunes parts):

```
1  public class Calculadora {
2
3      private float lastResult;
4      private String lastOp;
5
6      public float getLastResult(){ ... }
7
8      ...
9
10     public float suma(float op1, float op2){ ... }
11     }
12 }
```

- Com veiem, la classe es defineix amb **`public class`**, indicant que es tracta d'una classe pública (es pot utilitzar des de fora).
- Dins la classe hem definit:
  - Atributs, com `lastResult` i `lastOp`, que hem definit com a **`private`**. El primer és un número en coma flotant (**`float`**) i el segon una cadena de caràcters (`String`). El fet de definir-los com a *privats* significa que directament no es pot accedir a aquests atributs des de fora de la classe. A grans trets, els atributs privats d'una classe podríen interpretar-se com variable globals només a la classe. Aquests atributs, el que fan és emmagatzemar el resultat de la última operació realitzada i quina ha estat aquesta.
  - Mètodes (funcions), com `getLastResult`, `getLastOp`, que es coneixen com a *getters*, i són mètodes a través dels quals podem consultar els valors dels atributs privats. També podem definir *setters*, que estableixen aquests valors, però no són necessaris a l'exemple. Aquests es defineixen com a **`public`**, per tal de poder utilitzar-los des de fora de la classe.
  - Mètodes (funcions) com `suma`, `resta`, `multiplica` i `divideix` que són els qui realitzen les diferents operacions, guardant el resultat i l'operació i retornant el resultat. Dins aquests mètodes veurem que apareix la paraula reservada **`this`**. Aquesta s'utilitza per fer referència al propi objecte. Així, per exemple, **`this`**`.lastOp` fa referència a l'atribut `lastOp` del propi objecte.

El codi font de l'aplicació està bastant explicat, pel que poden consultar en ell la resta de mètodes. Tingueu en compte que aquest és un exemple introductori a Java, i que hi ha molt més contingut pel que respecta a la programació orientada a objectes.

Classe com.ieseljust.edd.calc.Calcula

Aquesta és la classe que implementa la funcionalitat principal (el mètode `main`), i fa ús de la llibreria que hem creat amb la classe *Calculadora*.

El fitxer corresponent a la classe és el `com/ieseljust/edd/calc/Calcula.java`:

```
1  package com.ieseljust.edd.calc;
2
3  import com.ieseljust.edd.calc.Calculadora;
4
5  public class Calcula {
6      private static float operand1;
7      private static float operand2;
8
9      public static void main(String[] args) {
10         if (args.length != 2) {
11             System.out.println("\nSintaxi incorrecta. Necessite dos nú
                   meros.");
12             System.exit(-1);
13         }
14
15         operand1=Float.parseFloat(args[0]);
16         operand2=Float.parseFloat(args[1]);
17
18         Calculadora myCalc=new Calculadora();
19
20         System.out.println("La suma entre "+operand1+" i "+operand2+" é
               s "+myCalc.suma(operand1, operand2));
21         System.out.println("La resta entre "+operand1+" i "+operand2+"
               és "+myCalc.resta(operand1, operand2));
22         System.out.println("La multiplicació entre "+operand1+" i "+
               operand2+" és "+myCalc.multiplica(operand1, operand2));
23         System.out.println("La divisió entre "+operand1+" i "+operand2+
               " és "+myCalc.divideix(operand1, operand2));
24         System.out.println("Última operació realitzada: "+myCalc.
               getLastOp()+"; Últim resultat: "+myCalc.getLastResult());
25     }
26 }
```

Analitzem el codi per parts.

- En primer lloc, definim el nom del paquet al que pertany el codi (`package com.ieseljust.edd.calc;`) i importem la llibreria *Calculadora* (`import com.ieseljust.edd.calc.`

`Calculadora;`).

- Després definim la classe pública `Calcula` que contindrà dos atributs de tipus float: `operand1` i `operand2`. Com veiem al codi, aquests atributs s'han definit com a estàtics **`static`**. Açò significa que són atributs propis de la classe (no prenen valor en cada instància que generem a partir d'ella). A efectes pràctics, açò ens serveis per poder utilitzar el *`main`* dins la classe sense necessitat de crear un objecte d'aquesta. Aquests atributs ens serviran per emmagatzemar els operands sobre els què es realitzaran les operacions.
- Aquesta classe només té un mètode, el *`main`*. Dins aquest mètode fem el següent:
  - En primer lloc, i mitjançant el vector d'arguments `args` que rep el `main`, comprovem, que li passem dos paràmetres al programa (en la invocació d'aquest des de la línia d'ordres). Amb aquest vector, donem valor als atributs `operand1` i `operand2`.
  - Per tal de poder utilitzar la classe *Calculadora*, hem de crear un objecte d'aquest tipus. Açò ho farem mitjançant l'operador *new*, i definint l'objecte com si definirem una variable del tipus de la classe: `Calculadora myCalc=new Calculadora();`. En aquest cas, hem definit l'objecte `myCalc` de tipus `Calculadora` i li hem assignat el resultat de la creació d'un *nou* (**`new`**) objecte `Calculadora`. Ara, mitjançant `MyCalc` ja podem accedir a les funcionalitats que aquesta llibreria ens proporciona.
  - Finalment, mostrem els resultats, accedint als valors emmagatzemats d'`operand1` i `operand2`, en aquest cas, sense **`this`**, ja que es tracta d'atributs de classe, i invocant als diferents mètodes de la llibreria a través de l'objecte *myCalc*; per exemple `myCalc.suma(operand1, operand2)`.

Compilació i execució

Per tal de compilar directament el nostre programa, podem fer:

1. Compilar de forma individual els dos fitxers:

```
$ javac com/ieseljust/edd/calc/Calculadora.java
$ javac com/ieseljust/edd/calc/Calcula.java
```

2. Compilar directament tots els fitxers `.java` del directori calc:

$ javac com/ieseljust/edd/calc/*.java

De tota manera, `javac` és suficientment intel·ligent com per saber que si compile `Calcula.java` que importa la classe *Calculadora*, ha de compilar aquesta també si no existeix el fitxer **`.class`** corresponent.

Una vegada generats els fitxers de bytecode, executem la calculadora amb:

```
$ java com.ieseljust.edd.calc.Calcula num1 num2
```

Reemplaçant *num1* i *num2* pels números que desitgem.

> **TO-DO**
>
> A partir del codi proporcionat, implementeu la nova funcionalitat a la calculadora `MajorQue`, que indique si el primer argument que li proporcionem és major que el segon. El tipus de valor de retorn haurà de ser lògic.
>
> Feu ús d'aquest mètode en l'aplicació, escrivint, immediatament després d'escriure el resultat de la divisió, el resultat d'aquesta comparació.

2. Apache Ant (Another Neat Tool)

Apache Ant és una llibrería de Java que ens permet automatitzar el procés de construcció d'aplicacions. El seu ús principal ha estat Java, tot i que també es pot utilitzar per a aplicacions en altres llenguatges. Inicialment va ser part del projecte Apache Tomcat, però l'any 2000 es va llençar com a projecte independent.

2.1. El fitxer build.xml

El fitxer de construcció d'Ant està escrit en XML (generalment s'anomena build.xml), i conté diversos *targets*, que representen les diferents fases de construcció (semblant als targets del make).

Veiem un exemple per a la nostra calculadora:

```
<project>
    <target name="clean">
        <delete dir="classes" />
    </target>

    <target name="compile" depends="clean">
        <mkdir dir="classes" />
    <javac includeantruntime="false" srcdir="com/ieseljust/edd/calc"
       destdir="classes" />
    </target>

    <target name="run" depends="compile">
        <java classpath="classes" classname="com.ieseljust.edd.calc.
           Calcula">
            <arg value="${arg0}"/>
            <arg value="${arg1}"/>
        </java>
```

```
16
17      </target>
18  </project>
```

Nota: Aquest fitxer *build.xml* l'haurem de crear al directori arrel del codi de la nostra aplicació. Al cas de la calculadora, haurà d'estar al mateix directori on està el directori *com*:

```
calculadora
    |-- build.xml
    `-- com
        `-- ieseljust
            `-- edd
            ...
```

Com veiem, l'XML té tres etiquetes *target*:

- **clean**: que *neteja* les eixides anteriors de la construcció, concretament, eliminant la carpeta classes.
- **compile**: Que realitza la compilació de l'aplicació. Com veiem, depèn de *clean*, i el que fa és crear la carpeta classes, i invocar el compilador de java (javac). L'atribut *includeantruntime* s'afig per tal d'evitar un missatge d'alerta a partir de la versió 1.8.
- **run**: Per tal d'executar l'aplicació. Com veiem, depèn de la tasca de compilació, i el que fa és llançar java sobre la classe `com.ieseljust.edd.calc.Calcula` que es troba al classpath classes. A més, dins l'etiqueta que invoca la màquina virtual de Java (`<java>`), afegim dues etiquetes `<arg>`, que agafaran els arguments que passem per la línia d'ordres, amb les variables `arg0` i `arg1`.

2.2. Instal·lació d'ant

Per tal de comprovar el funcionament, cal primerament instal·lar el paquet ant. Per comprovar si el tenim instal·lat, farem:

```
$ apt-cache policy ant
ant:
  Instal·lat: (cap)
  Candidat:   1.10.5-3~18.04
  Taula de versió:
     1.10.5-3~18.04 500
        500 http://archive.ubuntu.com/ubuntu bionic-updates/universe
            amd64 Packages
        500 http://archive.ubuntu.com/ubuntu bionic-updates/universe
            i386 Packages
        500 http://security.ubuntu.com/ubuntu bionic-security/universe
            amd64 Packages
```

```
10          500 http://security.ubuntu.com/ubuntu bionic-security/universe
               i386 Packages
11      1.10.3-1 500
12          500 http://archive.ubuntu.com/ubuntu bionic/universe amd64
               Packages
13          500 http://archive.ubuntu.com/ubuntu bionic/universe i386
               Packages
```

Com veiem, ens mostra que *ant* no es troba instal·lat al sistema, però tenim una versió candidata als repositoris d'Ubuntu (la 1.10.5-3~18.04). En cas d'estar instal·lada, en lloc de `(cap)`, se'ns mostraria la versió d'aquesta.

Per tal d'instal·lar `ant`, farem:

```
1 $ sudo apt install ant
```

2.3. Ús d'Ant

Una vegada instal·lat, i amb el fitxer `build.xml` creat, podrem llençar **ant** amb els diferents targets. Veiem l'eixida de cadascun:

- Per tal de compilar el projecte, farem:

```
1  $ ant compile
2  Buildfile: /home/joamuran/Dropbox/Docencia/curs_19-20/EDD/Unitats/UD4.
      Automatitzacio/exemples_java/calcula/build.xml
3
4  clean:
5
6  compile:
7      [mkdir] Created dir: ./calcula/classes
8      [javac] Compiling 2 source files to /home/joamuran/Dropbox/Docencia
          /curs_19-20/EDD/Unitats/UD4. Automatitzacio/exemples_java/
          calcula/classes
9
10 BUILD SUCCESSFUL
11 Total time: 0 seconds
```

Com veiem, ens llença primerament la tasca clean. Si és la primera vegada que l'executem, com a l'exemple, veiem que no fa res, ja que no té res a netejar. Si ja tinguérem la carpeta *classes* creada d'altres compilacions, ara ens indicaría que ha esborrat aquesta carpeta.

Després de fer la neteja, fa la tasca de compilació, amb la que crea la carpeta classes, i invoca el compilador de java per generar el fitxer en bytecode (*.class*).

Una vegada feta la compilació, podem observar l'estructura generada:

```
.
|-- build.xml
|-- classes
|   `-- com
|       `-- ieseljust
|           `-- edd
|               `-- calc
|                   |-- Calcula.class
|                   `-- Calculadora.class
`-- com
    `-- ieseljust
        `-- edd
            `-- calc
                |-- Calculadora.java
                `-- Calcula.java
```

Com veiem, ens ha generat una carpeta *classes* tal i com hem indicat al target, i dins d'aquesta, ens ha replicat tota l'estructura de carpetes corresponent al paquet, amb els fitxers *.class* compilats a dins.

- Per a l'execució:

```
$ ant run -Darg0=3 -Darg1=4
Buildfile: /home/joamuran/Dropbox/Docencia/curs_19-20/EDD/Unitats/UD4.
   Automatitzacio/exemples_java/calcula/build.xml

clean:
   [delete] Deleting directory /home/joamuran/Dropbox/Docencia/curs_19
      -20/EDD/Unitats/UD4. Automatitzacio/exemples_java/calcula/classes

compile:
    [mkdir] Created dir: /home/joamuran/Dropbox/Docencia/curs_19-20/EDD
       /Unitats/UD4. Automatitzacio/exemples_java/calcula/classes
    [javac] Compiling 2 source files to /home/joamuran/Dropbox/Docencia
       /curs_19-20/EDD/Unitats/UD4. Automatitzacio/exemples_java/
       calcula/classes

run:
     [java] La suma entre 3.0 i 4.0 és 7.0
     [java] La resta entre 3.0 i 4.0 és -1.0
     [java] La multiplicació entre 3.0 i 4.0 és 12.0
     [java] La divisió entre 3.0 i 4.0 és 0.75
     [java] Última operació realitzada: Divideix; Últim resultat: 0.75

BUILD SUCCESSFUL
Total time: 1 second
```

Fixeu-vos que l'ordre `ant run` afig els dos arguments que necessita la nostra aplicació amb `-Darg0=3 -Darg1=4`. És a dir, posem `-D` seguit del nom de l'argument tal i com s'anomena al fitxer `build.xml`.

Com veiem, en aquest cas, com que no havíem llençat prèviament el `ant clean`, la tasca *clean*, sí que ha esborrat el directori classes. Posteriorment, ha fet la compilació, i finalment, l'execució.

Finalment, si volguérem netejar el projecte, fariem ús de `ant clean`.

> TO-DO
>
> Creeu el projecte *Ant* amb el vostre codi font i verifiqueu que aquest es neteja, es compila i s'executa correctament amb els arguments.

2.4. Projectes Ant amb Netbeans

Als apartats anteriors hem vist com crear un projecte *Ant* des de zero, i a través dels seus fitxers de configuració, fent ús de qualsevol editor de text.

L'IDE Netbeans ha utilitzat *Ant* de forma nativa quan hem generat algun projecte, de forma transparent a l'usuari. A les versions de Netbeans més recents, a banda d'*Ant*, també podem generar projectes amb *Maven* o *Gradle* entre d'altres. De tota manera, ens centrarem ara en projectes de tipus *Ant*.

Per a això, obrirem l'IDE Netbeans, i crearem un projecte nou amb *File -> New Project*, per tal d'obrir el diàleg de creació d'un projecte nou.

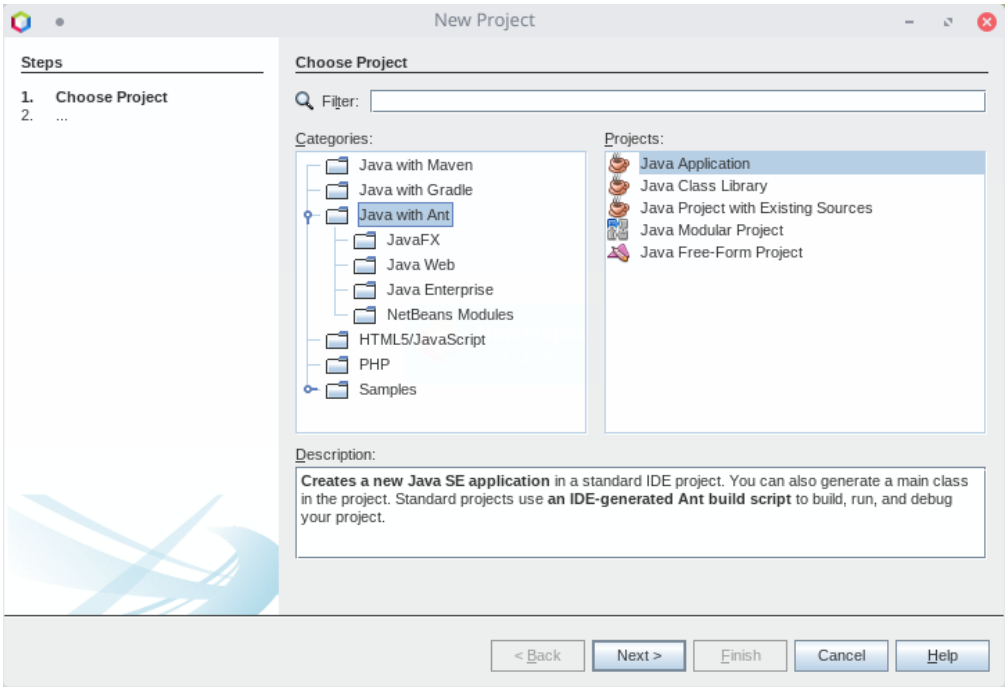

Figura 1: Creació d'un nou projecte

En aquesta finestra haurem d'indicar que volem una categoría de projecte *Ant* i que anem a fer es tractarà d'una *aplicació Java*, tal i com s'indica a la figura. Fem clic en *Next* quan ho tinguem seleccionat.

La següent finestra d'aquest assistent ens demanarà el nom del projecte, la seua ubicació i el nom de la classe principal. Indiqueu la ubicació on desitgeu guardar el projecte (a l'exemple es troba en `tmp`). El nom del projecte hem triat *Calculadora*, i com a classe principal, cal que indiqueu `com.ieseljust.edd.calc.Calcula`:

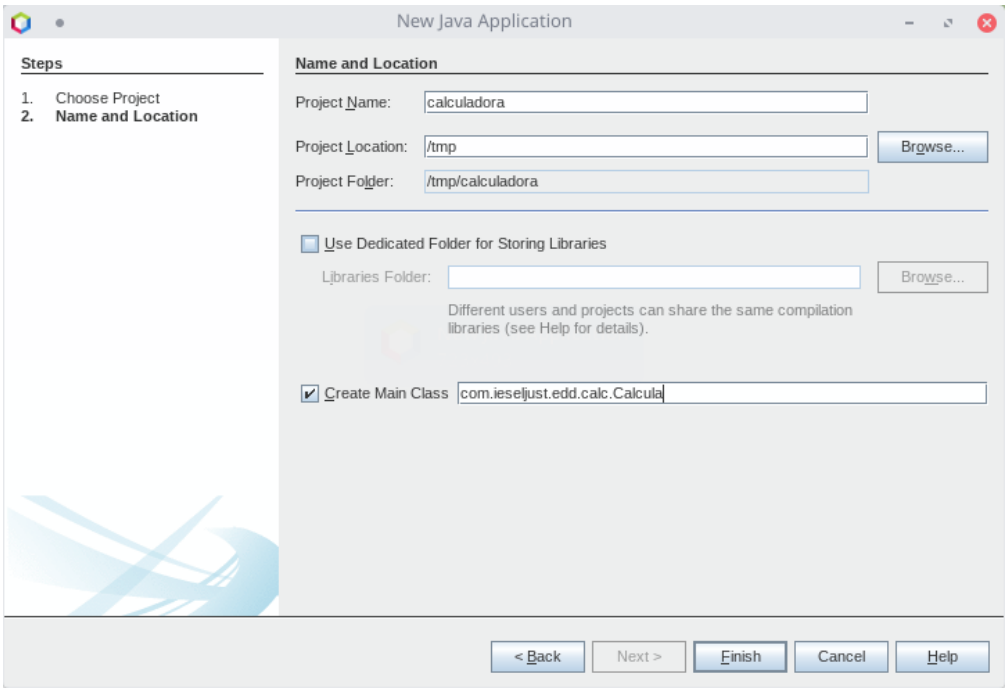

Figura 2: Informació del projecte

Quan fem clic en *Finish*, finalitzarem l'assistent i ens crearà automàticament el paquet font `com.ieseljust.edd.calc.Calcula` a la carpeta que li hajam indicat. En aquest punt, **copiarem tot el contingut del fitxer font `Calcula.java` en aquesta classe, que és la principal de l'aplicació i que és, per tant la que conté el *main*.

Una vegada apegat aquest codi principal, veurem que Netbeans ens indica (subratllant el **`import`**) que no troba la classe `com.ieseljust.edd.calc.Calculadora`, com és d'esperar, ja que aquesta classe encara no existeix. Per tal de crear la nova classe, farem clic sobre el recurs *Source Packages* del projecte, i seleccionarem *New -> Java Class*:

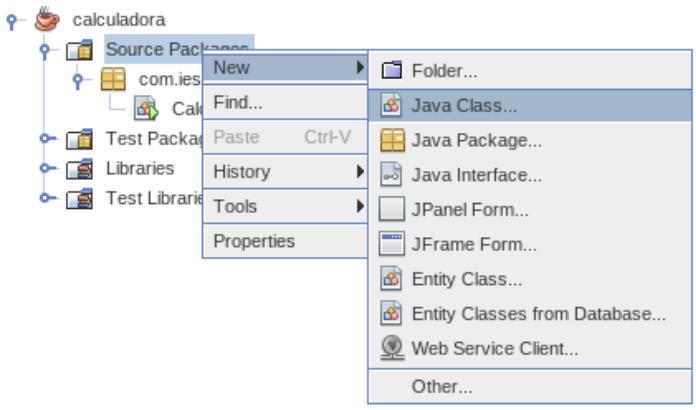

Figura 3: Creació d'una nova classe

Ens apareixerà un quadre de diàleg demanant-nos alguns detalls de la classe. Indiquem, respectant les majúscules i minúscules, el nom de la classe *Calculadora*, i que es trobarà al paquet `com.ieseljust.edd.calc`, tal i com s'indica a la figura:

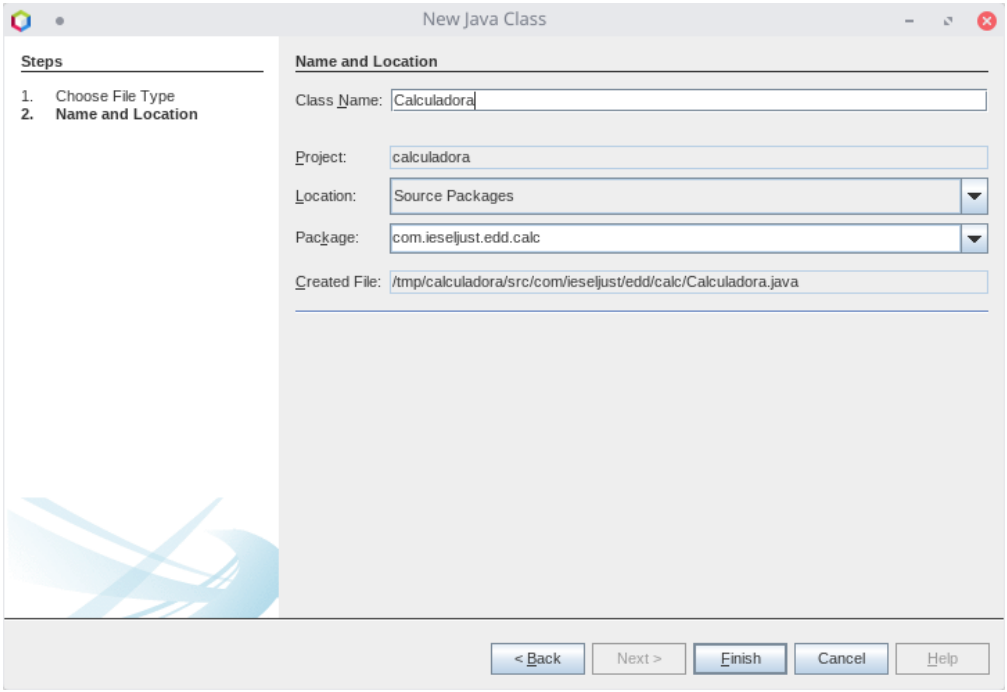

Figura 4: Detalls de la nova classe

Una vegada creada, apeguem sobre ella el contingut del fitxer *Calculadora.java* de l'exemple anterior, per tindre tot el codi de l'aplicació.

Ara, fixeu-vos en l'estructura del projecte que ens mostra Netbeans:

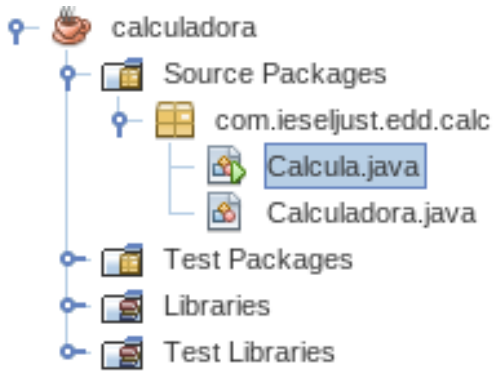

Figura 5: Estructura del projecte

Com veiem, tenim una carpeta *Source packages*, on dins ens apareix el paquet `com.ieseljust.edd.calc`, i dins d'ell els dos fitxers corresponents a les classes `Calcula.java` i `Calculadora.java`.

Com veiem, Netbeans ens ofereix una organització lògica del projecte, amb els paquets de fonts, tests... però... com s'organitza tot açò en carpetes? Si seleccionem a les pestanyes superiors del panell de l'esquerra la pestanya ***Files***, podrem veure com s'organitzen les carpetes d'aquest projecte:

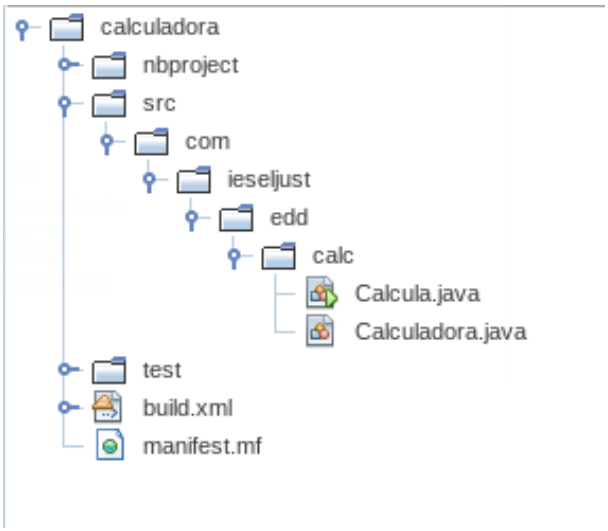

Figura 6: Estructura de carpetes

Com veiem, aquesta organització ja ens és més familiar. Veiem que ens ha generat tot el *path* corresponent a la ubicació dels paquets fonts (`com/ieseljust/edd/calc`), i que es troba a l'arrel del projecte també el fitxer *build.xml*. A més, Netbeans ens ha generat alguns recursos més, com la carpeta *NBProject* amb informació del projecte per a Netbeans, el fitxer de Manifest, amb informació també del projecte, i la carpeta *test* per als tests.

Ara obriu el fitxer *build.xml* i observeu-ne el seu contingut. Com voreu, tot i ser un fitxer més llarg, inclou menys *targets* que el nostre fitxer de construcció. En ell podem vore què apareixen molts comentaris d'ajuda per especificar els nostres targets. De moment, no anem a modificar aquest fitxer, ja que Netbenas ens deixa realitzar la compilació i l'execució directament. Només ens faltarà un xicotet detall, i és que com que necessitem passar-li arguments a l'aplicació haurem de modificar les propietats del projecte. Per a això, fem clic al botó dret sobre el projecte, per tal d'obrir el següent menú contextual, i seleccionem *Properties*:

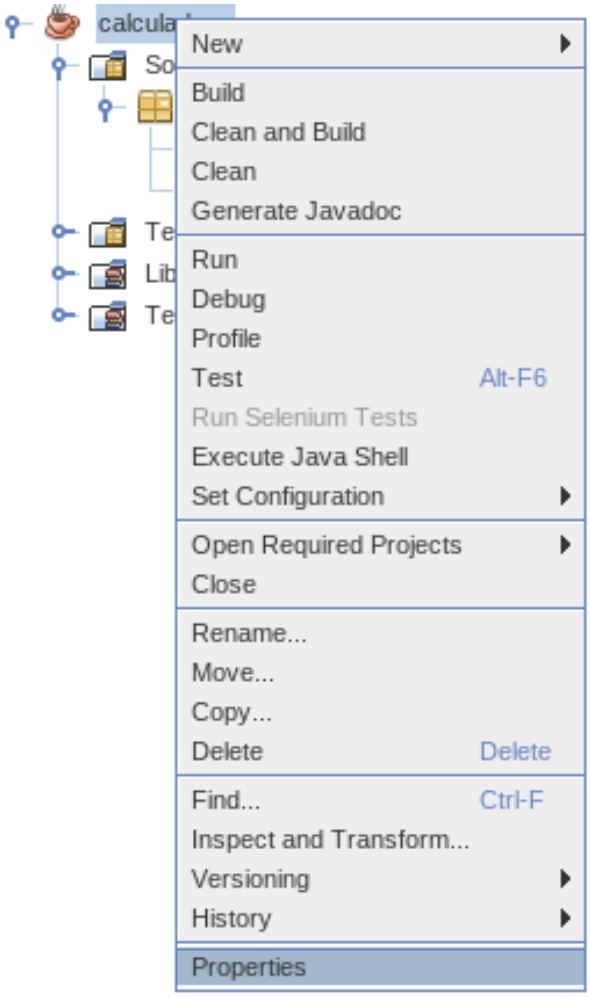

Figura 7: Accés a les propietats del projecte

En el diàleg que ens apareix, busquem la categoria *Run* i indiquem els *Arguments* que volem passar-li al programa. En aquest cas, hem posat directament els valors *3* i *5*:

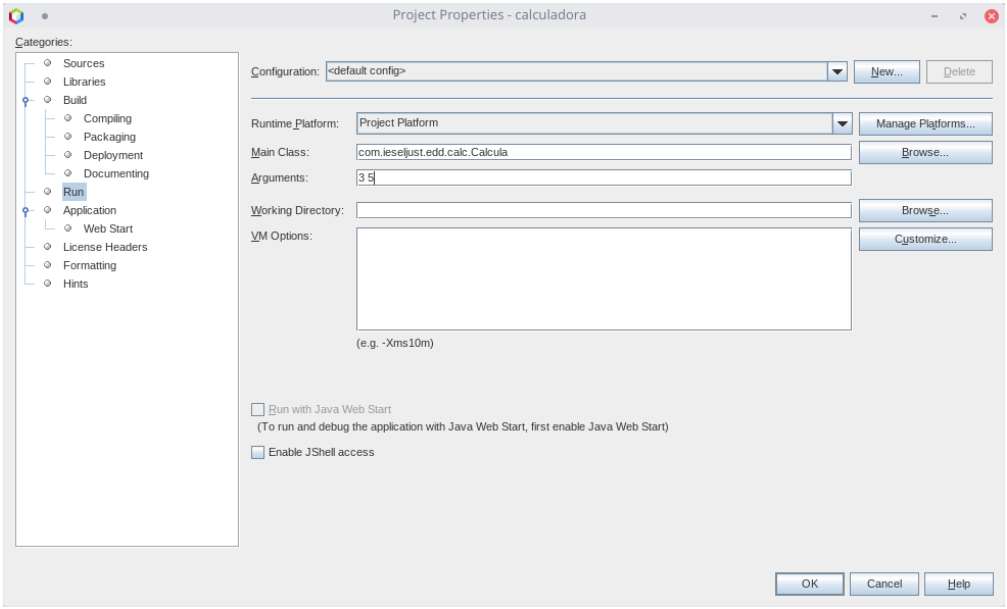

Figura 8: Propietats del projecte

Una vegada modificat, si fem clic al botó del *Play* que tenim a la barra superior, podem compilar i executar el projecte amb els arguments que hem proporcionat. Si volem modificar-los, haurem de fer-ho a través de les propietats del projecte.

> TO-DO
>
> Creeu ara el projecte de la calculadora amb Netbeans, amb el vostre codi font, i comproveu els resultats de l'execució.
>
> A mode de conclussió, reflexioneu (per escrit) sobre les següents qüestions: Quin dels dos mètodes (de forma manual o amb l'assistent de Netbeans) creieu que és millor? Quin us resulta més còmode d'utilitzar? Quin us ofereix més control sobre tot el què es crea?