

PAC 1

Començant...

UOC

Informació rellevant:

- Data límit de lliurament: 16/10/2023
- Pes a la nota d'AC: 10%.

Universitat Oberta
de Catalunya

Contingut

Informació docent	3
Prerequisits	3
Objectius	3
Resultats d'aprenentatge	3
Enunciat	4
Tasca 1 - Configuració de l'entorn bàsic	4
Tasca 2 - Configurant IntelliJ	6
Exercici 1 - Debugging (2 punts)	12
Tasca 3 - Testing amb JUnit	14
Exercici 2 (3 punts)	19
Exercici 3 (5 punts)	21
Format i data de lliurament	23

Informació docent

Aquesta PAC està vinculada al mòdul teòric “Introducció al paradigma de la programació orientada a objectes” dels apunts de l'assignatura. Llegeix-ho abans de començar la PAC.

Prerequisits

Per fer aquesta PAC necessites saber:

- Conceptes bàsics d'algorítmica (après a l'assignatura anterior).
 - Conèixer el paradigma de la programació estructurada.
 - La sintaxi d'un llenguatge imperatiu d'ús comú, p. C, Python, etc.
 - Declaració i ús de variables de tipus primitiu (`int`, `float`, etc.).
 - Bucles (`while`, `for` i `do-while`).
 - Condicionals (`if-else` i `switch`).
 - Ús d'arrays.
 - Creació i ús de funcions (en el nostre context anomenades “mètodes”).

Objectius

Amb aquesta PAC l'equip docent de l'assignatura busca que:

- Utilitzis Java com a llenguatge de programació imperatiu i, per tant, coneguis la sintaxi bàsica d'aquest llenguatge.
- Et familiaritzis amb la dinàmica de l'assignatura: metodologia, tipologia d'enunciats, nomenclatures, etc.

Resultats d'aprenentatge

Amb aquesta PAC has de demostrar que ets capaç de:

- Configurar un entorn de treball bàsic per programar amb Java: JDK i Notepad++.
- Implementar programes senzills a Java basats en el paradigma de la programació estructurada.
- Entendre què fa un programa o codi donat per un tercer.
- Configurar i utilitzar amb solvència un entorn de desenvolupament integrat (IDE) com IntelliJ.
- Ser capaç de depurar (*depurar*) en un IDE un programa amb errors per corregir-ho.
- Utilitzar tests unitaris per determinar que un programa és correcte.

Enunciat

Aquesta PAC conté 3 tasques (activitats no avaluables) i 3 exercicis (sí avaluables). Has de lliurar la teva solució dels 3 exercicis avaluables (veure el darrer apartat).



Com que les activitats estan encadenades (i.e. per fer-ne una s'ha d'haver comprès l'anterior), **és altament recomanable fer les tasques i els exercicis en l'ordre en què apareixen en aquest enunciat.**

Tasca 1 - Configuració de l'entorn bàsic

Abans de començar has de:

- Instal·lar el JDK al teu ordinador. Llegiu l'apartat "1.1. Instal·lant JDK" de la Guia de Java. **Recomanem instal·lar JDK 17.** A més, es recomana llegir l'apartat 1.2 de la Guia de Java.
- Usar un editor de text pla (p.ex. el que ve amb el teu sistema operatiu): vim, notepad, TextEdit, etc. No obstant això, per a Windows recomanem [Notepad++](#).

Un cop realitzats els passos anteriors, pots fer la tasca següent per comprovar que l'entorn bàsic per desenvolupar programes en Java el tens ben instal·lat. Per això, segueix aquests passos:

1. Obre el directori `PAC1Task1` que et proporcionem amb l'enunciat.
2. Allà trobaràs un fitxer anomenat `PAC1Task1.java`.
3. Obre un terminal o processador de comandes (cmd).
4. Al terminal ubica't al directori `PAC1Task1` i un cop executa la comanda:

```
javac PAC1Task1.java
```

5. Si tot va bé, al terminal no passarà absolutament res. Això vol dir que el fitxer `PAC1Task1.java` no ha produït cap error. Tot i això, si mires dins del directori `PAC1Task1`, hauria d'haver aparegut un nou fitxer anomenat `PAC1Task1.class`. Aquest nou fitxer és el *bytecode* de `PAC1Task1.java`.
6. A continuació executarem `PAC1Task1.class`. Per fer-ho, executa la següent ordre al terminal que tens obert (que està ubicat al directori `PAC1Task1`):

```
java PAC1Task1
```

7. Veureu que el programa escriu el missatge següent: No data

8. Ara executa el programa `PAC1Task1` passant-li arguments. Per exemple:

```
java PAC1Task1 5 -4 12 3 -20
```

9. L'execució anterior hauria de produir la sortida següent per pantalla:

```
-20 12 -0,8
```

10. Fes més proves passant diferents arguments. Després obre el fitxer `PAC1Task1.java` i observa el codi que conté. Així mateix, prova altres execucions del programa. Què creus que fa aquest programa?

11. Resposta:

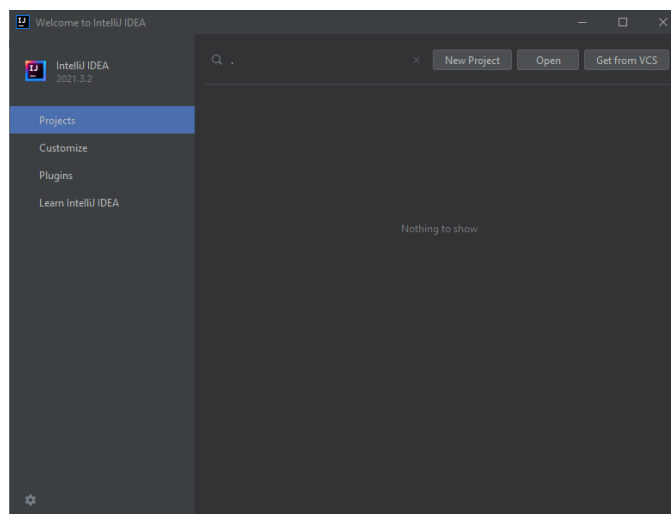
Aquest mètode mostra per pantalla, en funció del llistat de números passats com a paràmetre, (1) el valor mínim del llistat, (2) el valor màxim, i (3) el valor mitjà. Si no es facilita cap número, es mostra el missatge "No data".

Tasca 2 - Configurant IntelliJ

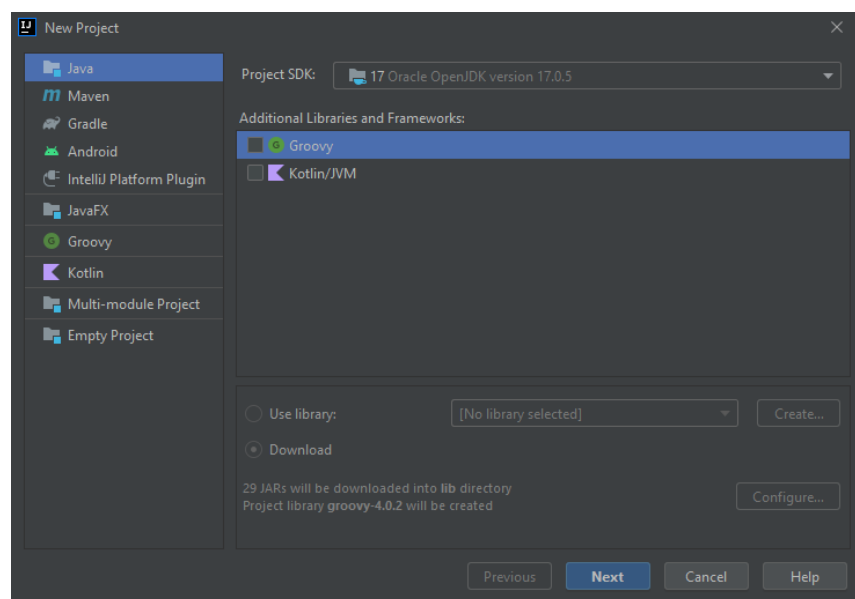
Abans de continuar has de:

Instal·lar i configurar l'IDE IntelliJ al vostre ordinador. Per això llegeix el document **Guia_IntelliJ_CAT.pdf** que trobaràs a l'activitat de l'aula.

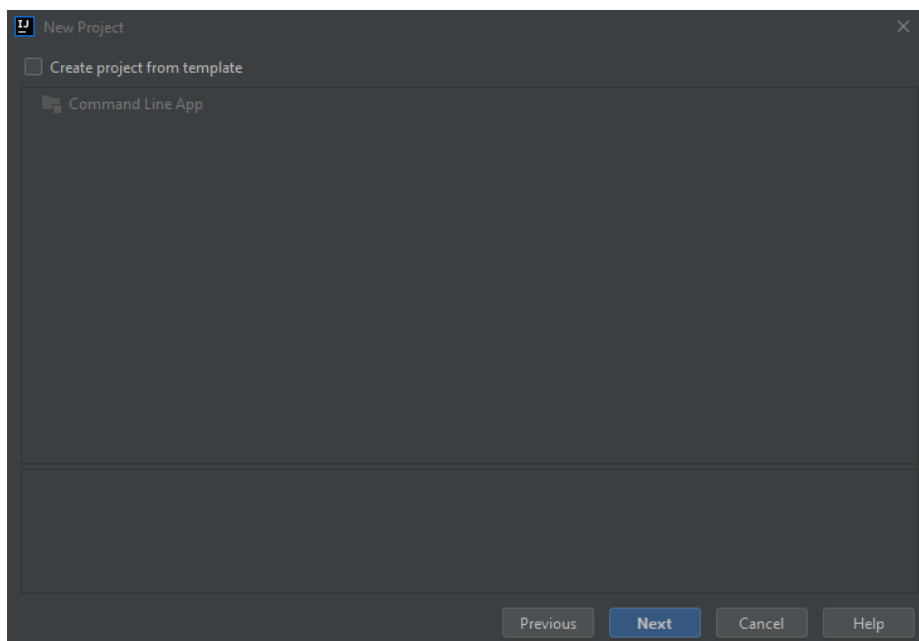
Un cop realitzat el pas anterior, **executa IntelliJ i crea un nou projecte** amb el botó **New Project**:



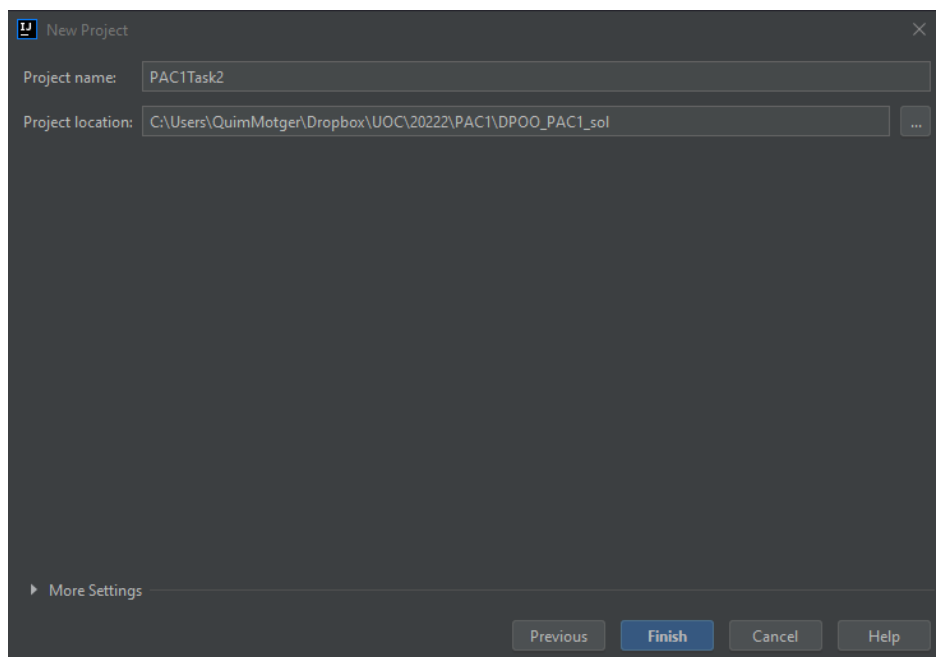
Un cop clicat el botó per crear un nou projecte, IntelliJ us mostrarà una finestra de configuració del projecte. Per defecte, IntelliJ selecciona com a tipus de projecte **Java** (llenguatge que farem servir durant tota l'assignatura) i com a SDK (*Kit de desenvolupament de programari*) el JDK que tens instal·lat a la teva màquina (JDK 17). Cliquem el botó **Next**.



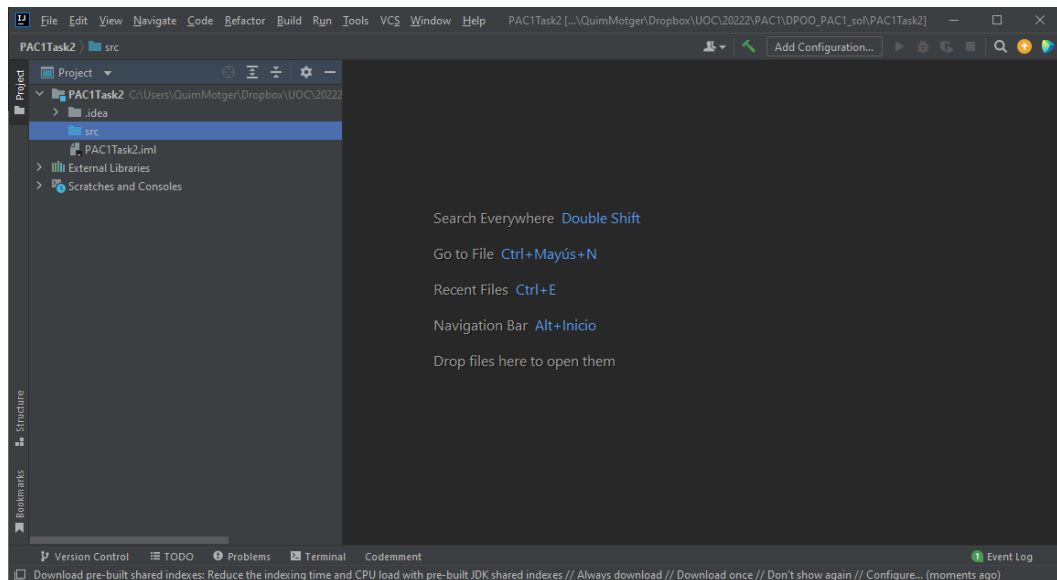
A la següent pantalla clica el botó `Next`, ja que no farem servir cap plantilla predefinida:



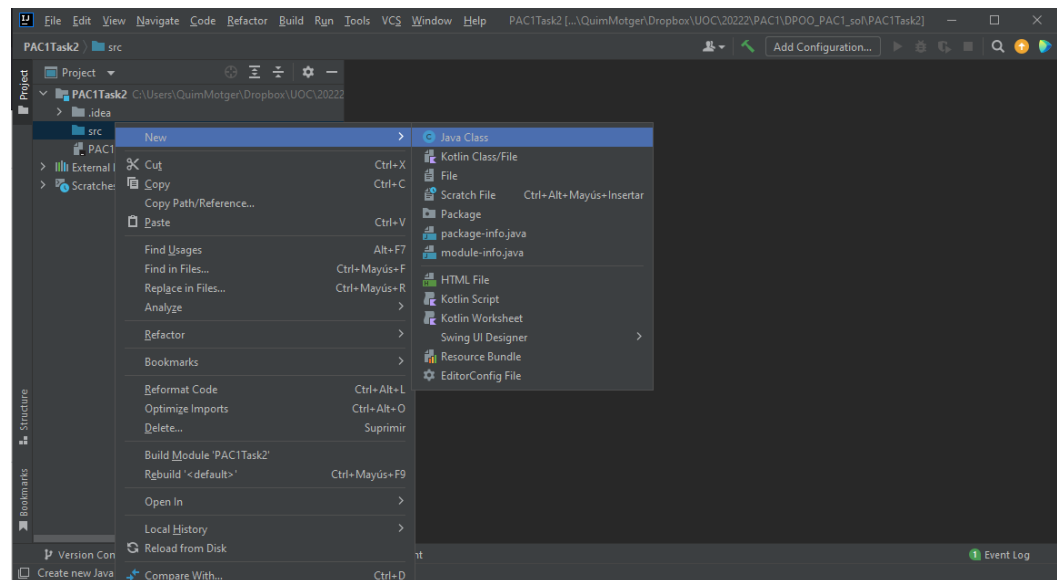
Finalment, introduïu el nom del projecte, en aquest cas, **PAC1Task2**, i selecciona la carpeta que farà de *workspace* (i.e. espai de treball) dels teus projectes. IntelliJ ja us aconsella un directori i un nom per al vostre espai de treball (i.e. `IdeaProjects`). Clica al botó `Finish` per crear aquest projecte Java.



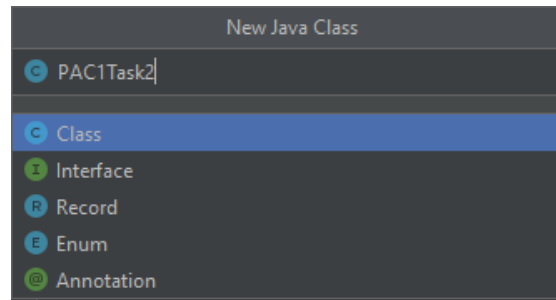
Un cop creat el projecte **PAC1Task2**, s'obrirà el projecte mostrant la *project tool window* al lateral esquerre. La *project tool window* mostra un arbre de directori del projecte. Hi veuràs la carpeta `src` buida. Aquesta carpeta/directori serà on posaràs el codi font del teu programa.



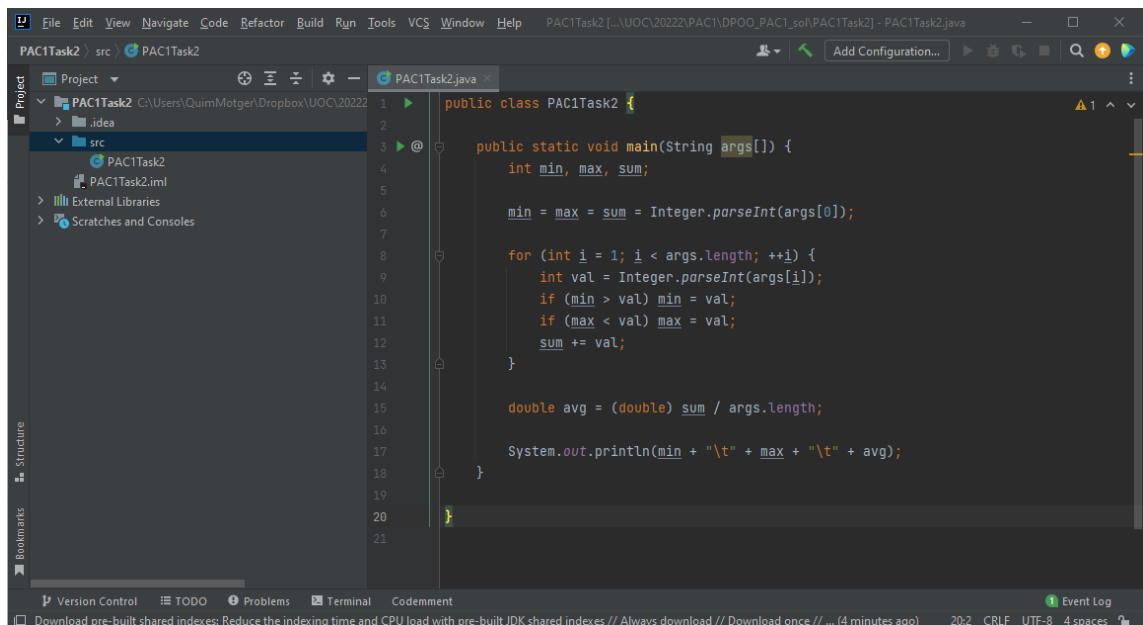
Fes clic amb el botó dret a sobre de la carpeta `src` i tria l'opció: `src` → New → Class Java.



A la finestra que apareixerà, introduïu el nom de la classe, en aquest cas, **PAC1Task2** (no cal posar-hi l'extensió `.java`, IntelliJ la posarà per tu).



Un cop creada la classe, copia al fitxer `PAC1Task2.java` tot el codi del mètode `main` (capçalera inclosa) del fitxer anomenat `PAC1Task1.java` (pots obrir-lo amb Notepad++) que trobaràs a la carpeta `PAC1Task1` que t'hem proporcionat amb l'enunciat i que has fet servir en la tasca anterior. Hauria de quedar-te una cosa semblant a això:

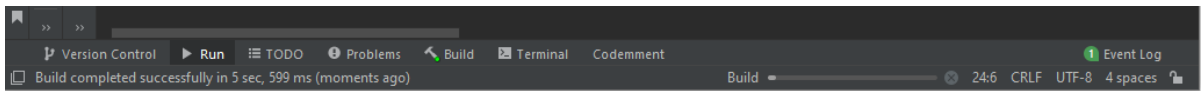


Truc: Si el codi t'apareix mal indentat (és a dir, mal formatat), pots fer `Ctrl+Alt+L` (Windows) o `Cmd+Alt+L` (MacOS) perquè IntelliJ indenti el codi correctament.

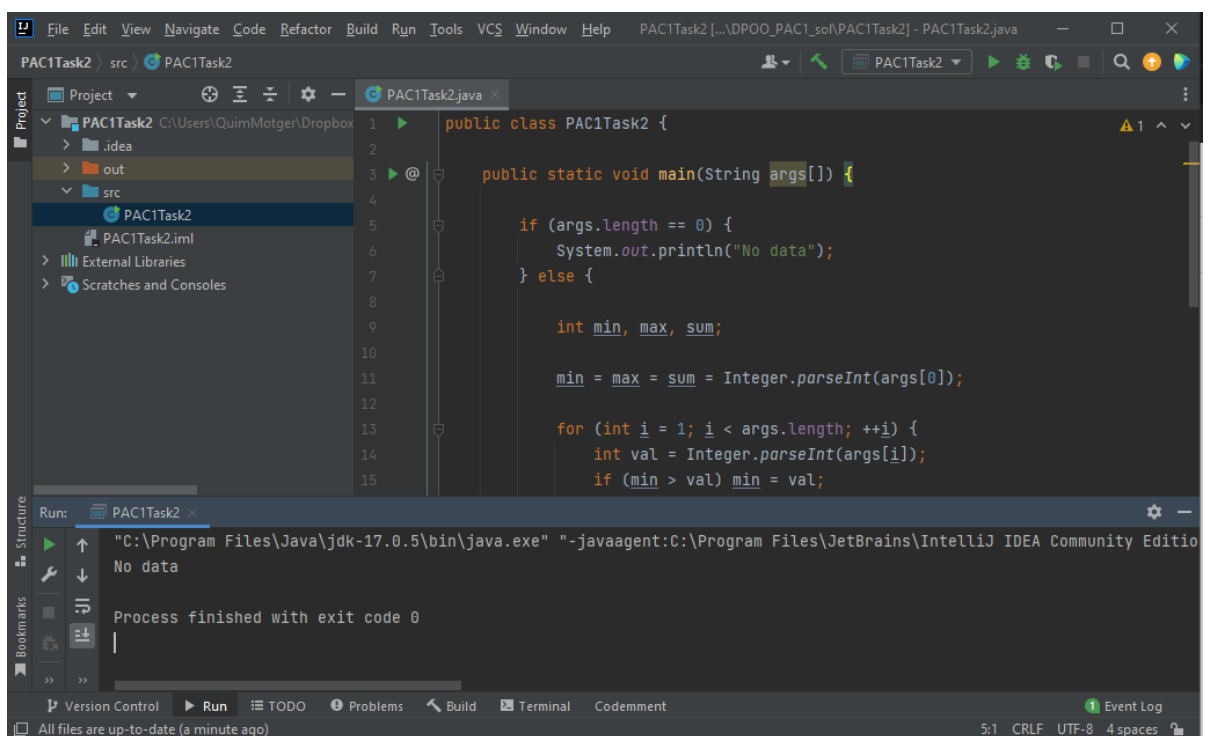
Si vols saber més dreceres (*shortcuts*) d'IntelliJ, fes un cop d'ull a: <https://www.jrebel.com/system/files/cheat-sheet-rebel-intellij-idea-web.pdf>.


Ara, al *project tool window*, selecciona el fitxer que acabes de crear `PAC1Task2.java` (l'extensió `.java` no apareix visible al *project tool window*), i fes clic amb el botó dret del ratolí. Al menú contextual que apareixerà escull l'opció: `Run "PAC1Task2.main()"`.

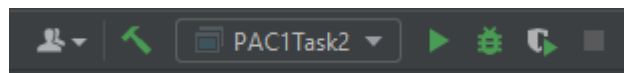
El programa serà compilat (la primera vegada pot trigar una mica) i, si no hi ha errors, s'executarà. Fixa't a la part inferior dreta d'IntelliJ per veure què està fent l'IDE amb el teu projecte. Indicarà l'estat en què està, p.ex. *"Build"*, *"Parsing"*, *"Build completed"*, etc.



Un cop executat t'ha d'aparèixer la consola amb el resultat *"No data"*, ja que el programa no rep cap llistat de números. Ara el codi de la Tasca 1 ho hem transformat en un projecte IntelliJ.



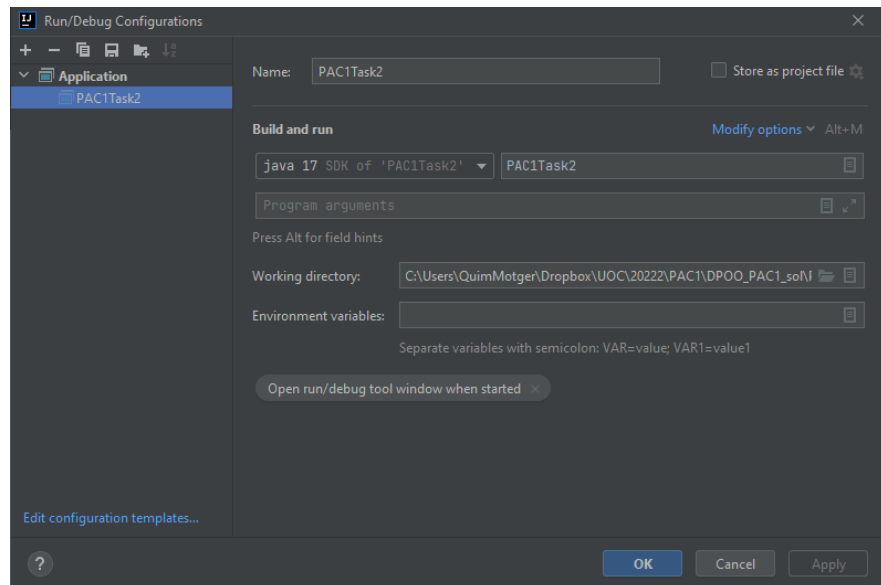
A continuació, fixa't que a la part superior dreta apareix la informació de la darrera execució que hem llançat. Ara podeu llançar l'execució directament des del botó **Run** .



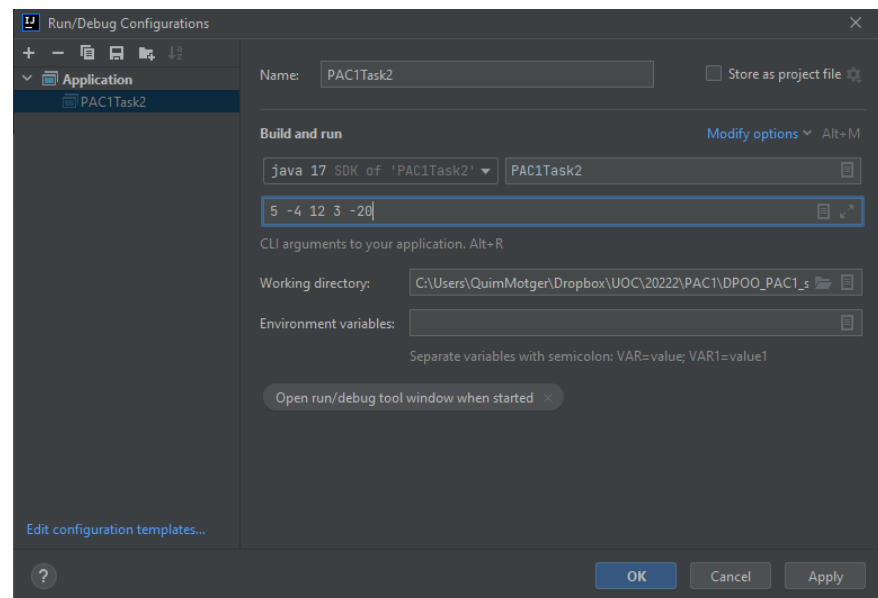
Si sempre dones a **Run**, sempre t'aparèixerà *"No data"*. És normal, ja que no li estàs passant arguments. Com passar arguments a l'execució?


1. Clica al desplegable que hi ha al costat del botó **Run** en què posa *PAC1Task2* i veuràs que desplega un menú.
2. Feu clic a l'opció *Edit configurations...*

3. S'obrirà una finestra com la que es mostra a continuació.



4. A la caixa de text que posa `Program arguments` escriu uns números separats per espais.



5. Si ara fas clic al botó **OK** de la finestra i, posteriorment, al botó **Run**  de la barra d'execució, s'executarà amb els valors que has escrit com a arguments a `Program arguments`, sent el resultat:

-20 12 -0,8

(si heu posat els números de l'exemple). Si vols modificar-los, has d'anar a `Edit configurations...` i repetir el procés.

Exercici 1 - Debugging (2 punts)

Per fer aquest exercici copia la carpeta **PAC1Ex1** que t'hem proporcionat dins la teva carpeta **workspace** (normalment anomenada **IdeaProjects**). Si obres la carpeta/projecte **PAC1Ex1** directament amb IntelliJ sense copiar-la al **workspace**, aleshores els canvis es veuran reflectits en el projecte que t'hem donat (pots fer-ho, però està bé tenir una còpia de seguretat i, alhora, saber que tots els teus lliuraments estan dins del **workspace**).

Un cop obert veuràs que apareixen errors. Els errors que IntelliJ mostra, o bé són sintàctics, o bé fan referència a elements que l'IDE no coneix (aquests errors també entren dins de l'apartat sintàctic perquè per a IntelliJ són com si estiguessin mal escrits). Un error sintàctic també pot ser l'absència o repetició d'una clau {}, o que l'ordre de les paraules clau i identificadors no està bé (p. ex. `myName int;`), etc.

Aquest programa, donat un nombre enter positiu n que introduïm a través del teclat, mostra per pantalla els n primers valors dels nombres pentagonals. Els nombres pentagonals son una successió infinita que expandeix el concepte de nombres triangulars i quadrats al pentàgon, però, a diferència d'aquests, els patrons que es fan servir en la seva construcció no son simètricament rotacionals. Aquesta successió comença amb els valor 1 (posició $n = 1$) i a partir d'aquests elements, per a qualsevol número consecutiu (posició $n > 1$), el valor corresponent es calcula mitjançant la següent formula:

$$P_n = n(3n - 1)/2$$

Per exemple: per $n = 8$, el programa mostra els 8 primers valors del nombres pentagonals:

posició	1	2	3	4	5	6	7	8
valor	1	5	12	22	35	51	70	92

- a) Corregeix, amb l'ajuda d'IntelliJ, els errors sintàctics del programa de manera que no en contingui cap.

(1 punt)

Ara si executes el programa veuràs que el resultat no és l'esperat. Per què? Perquè conté errors de lògica, és a dir, la funcionalitat no està ben programada. Evidentment IntelliJ no és màgic i no sap què pretens programar. Així doncs, has de comprovar mitjançant "traces" (i.e. executar el programa pas a pas) què està fent el programa i detectar on són els problemes. Per això pots, o bé posar en certs punts del programa instruccions que imprimeixin textos per pantalla (p.ex. `println`), o bé utilitzar la funcionalitat de debugging (en català, depuració) que ofereix IntelliJ (veure el videotutorial que trobaràs al [site](https://www.youtube.com/watch?v=bb7Ny3M6cpY) de l'assignatura (**UOCoders**), o el vídeo <https://www.youtube.com/watch?v=bb7Ny3M6cpY> i el tutorial: <https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html>).

Et recomanem que li dediquis una mica de temps a aprendre a debugar mínimament amb IntelliJ, ja que et serà molt útil en futures activitats (i al món professional).

- b) Corregeix els errors lògics (també anomenats errors semàntics) del programa de manera que s'executi com s'espera.

(1 punt)



Requisit mínim per avaluar aquest exercici: el programa no ha de contenir errors sintàctics (apartat A) ni semàntics (apartat B).

Tasca 3 - Testing amb JUnit

A l'Exercici 1 l'única manera que has tingut per saber si el programa es comportava com era esperat ha estat executant-lo diverses vegades i veient la sortida per pantalla (o pel debugger). Fer test i detectar errors utilitzant el mètode del propi llenguatge que escriu per pantalla (p.ex. `printf`, `print`, `println` o l'equivalent del llenguatge que utilitzis habitualment) és una manera molt bàsica –molt de principiant–, encara que tothom la sol utilitzar en algun moment per la comoditat. Per la seva banda, la debugació es fa servir per detectar errors locals dins del codi.

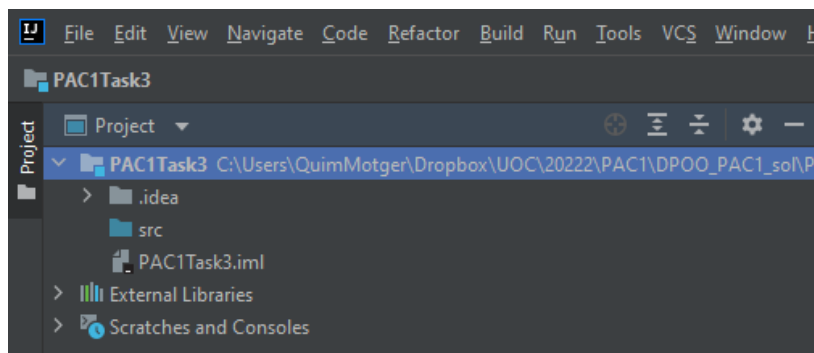
Atès que el *testing* és molt important per garantir que un programa és correcte i es comporta com s'espera, amb el temps ha sorgit una manera de desenvolupar programari basada justament en el testeig. És el que es coneix com a TDD o *Test Driven Development* (i.e. desenvolupament conduït/dirigit per test). Aquesta forma de desenvolupar té com a lema: *“un programador ha d'escriure els casos de test abans que el codi que aquests tests comprovaran”*.

A partir d'ara, en aquesta assignatura escriuràs o et proporcionarem test per a tots els exercicis que requereixen codificació. Per això farem servir la llibreria JUnit. És per això que abans de continuar has de:

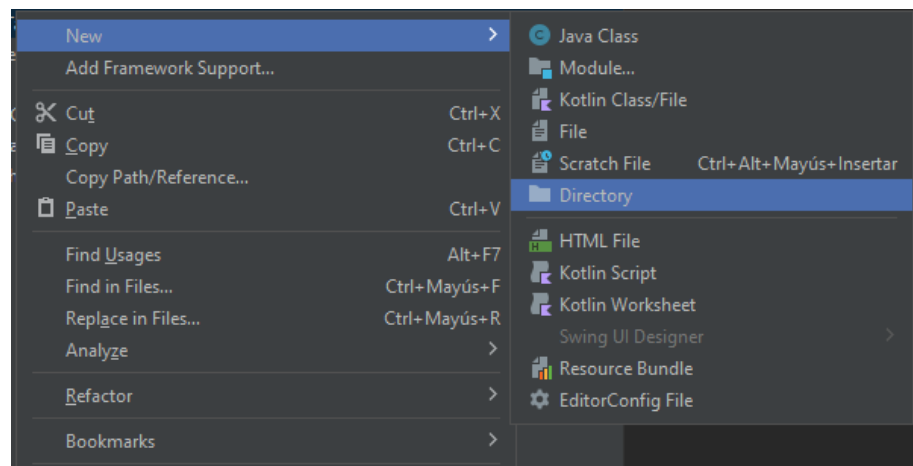
Llegir l'apartat **“5.3.1. JUnit”** de la Guia de Java, que dóna una petita introducció sobre què són els tests unitaris.

I per altra banda, seguir els passos següents per afegir, configurar i utilitzar JUnit en un projecte Java a l'IDE IntelliJ (sense cap suport d'un gestor de dependències com Maven o Gradle... això ja ho treballarem a la PAC2).

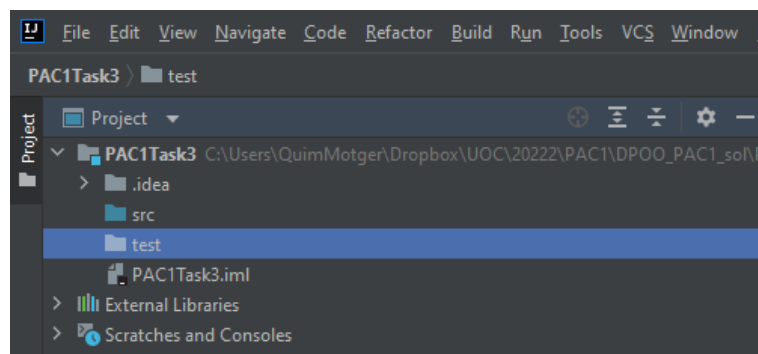
1. Crea un nou projecte Java a IntelliJ des de `File → New → Project` anomenat **PAC1Task3**.



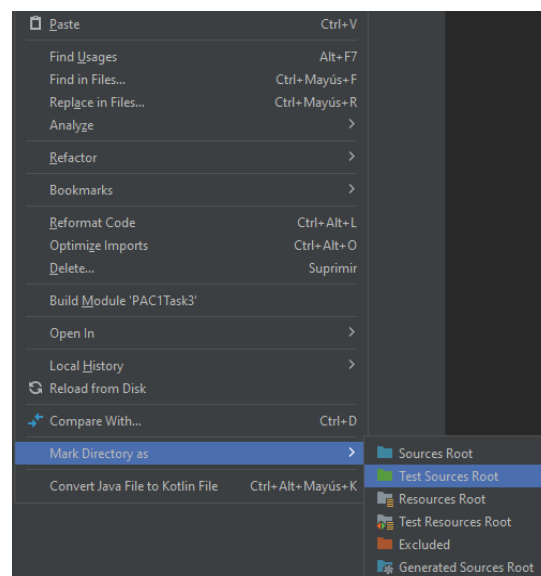
2. Des de l'arrel del projecte crea, amb el menú contextual que apareix en fer clic al botó dret del ratolí, un directori (`Directory`) anomenat `test` per emmagatzemar els tests unitaris JUnit.



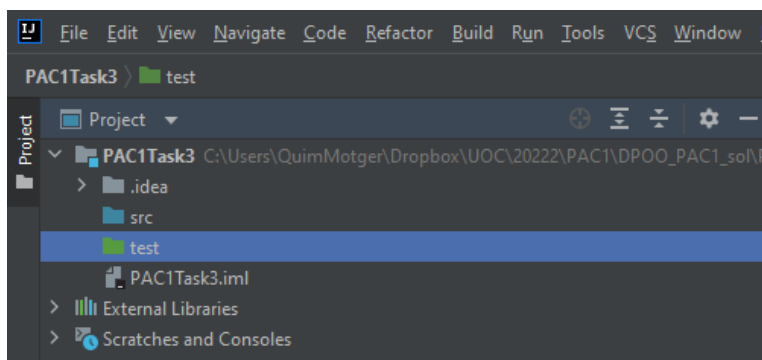
3. Un cop creat, veuràs al *project tool window* una nova carpeta `test`, al mateix nivell que la carpeta de codi font `src`.



4. Seleccioneu la carpeta `test` i marca-la com un directori que serà l'arrel de les fonts dels tests unitaris (i.e., *Test Sources Root*). Per fer-ho fes botó dret sobre el directori `test` i tria l'opció *Mark Directory as* → *Test Sources Root*.



Veureu que ha canviat el color de la carpeta `test` al verd.

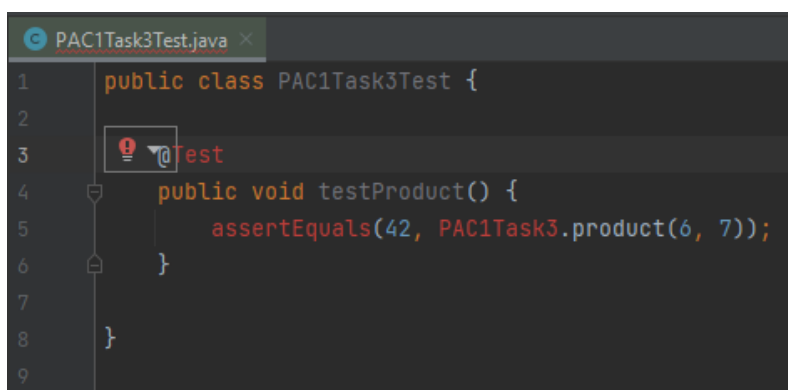



5. Un cop aquí, al directori `test`, crea la classe `PAC1Task3Test` on implementaràs els tests unitaris. Ja has vist anteriorment com crear classes als projectes.
6. Crea la funció/mètode següent dins d'aquesta classe:

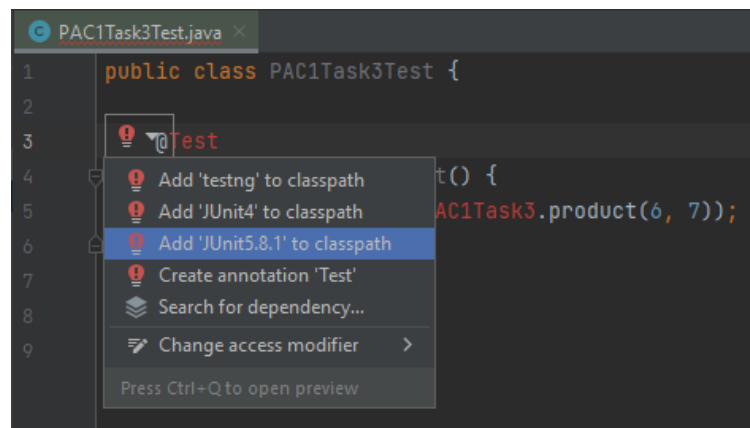
```
public void testProduct() {
    assertEquals(42, PAC1Task3.product(6, 7));
}
```

7. Afegeix l'annotació `@Test` damunt de la signatura del mètode. Amb això estem especificant que és un mètode de test JUnit. A més, això ens permet importar des de IntelliJ la llibreria JUnit.

Com veuràs, si poses el cursor a sobre de la línia `@Test`, us marca un error d'IntelliJ.

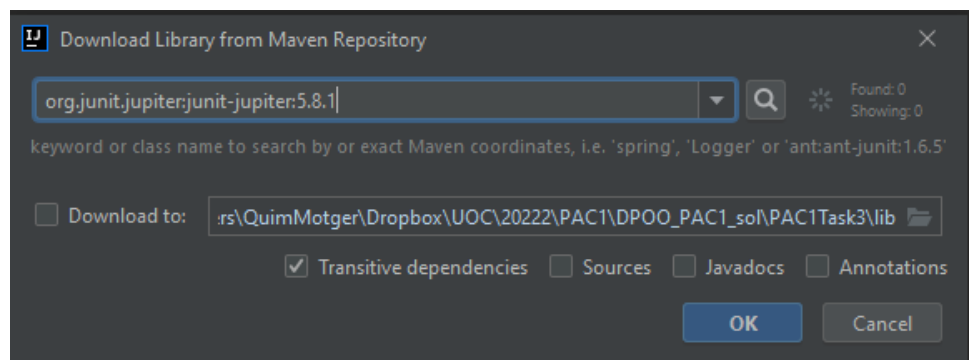


8. Clica sobre aquest globus vermell  i IntelliJ us proporcionarà una manera fàcil d'afegir la llibreria JUnit 5 al vostre projecte.



9. A la següent pantalla, IntelliJ et permet descarregar des del repositori Maven la llibreria JUnit 5.x (això no significa que utilitzis Maven com a gestor de dependències, únicament estàs dient que descarregueu aquesta llibreria des del repositori oficial de Maven; podria ser des d'un altre repositori o link).

Pots seleccionar el *check* Download to si vols descarregar els fitxers de la llibreria en una carpeta (lib) del teu projecte, o no marcar-lo si vols afegir la llibreria com External libraries. En aquest cas farem servir la segona opció.




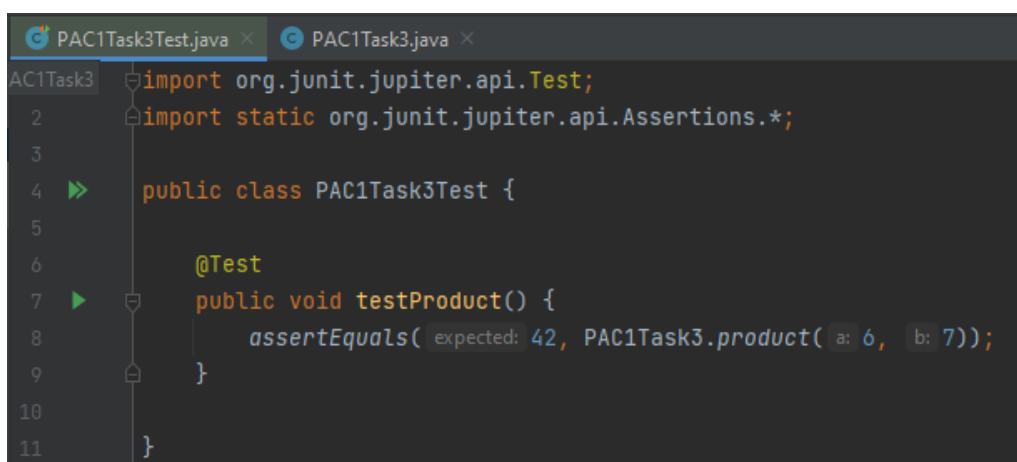
10. Un cop afegides les llibreries JUnit, veuràs que ja no tens l'error a l'anotació @Test.
11. Ara, al directori src, crea la classe PAC1Task3 on tindràs la implementació del mètode testProduct que has codificat a la classe de test PAC1Task3Test.
12. Dins el fitxer PAC1Task3.java escriu el codi següent:

```
public class PAC1Task3 {
    public static int product(int a, int b){
        return a * b;
    }
}
```

13. Finalment, ja només et queda resoldre l'error que hi ha a la classe `PAC1Task3Test`, ja que falta un `import` per poder utilitzar les *asserts* (i.e. mètodes/funcions) de JUnit. Per tant afegeix aquest `import` damunt de la definició de la classe (si no ho ha fet IntelliJ ja per tu):

```
import static org.junit.jupiter.api.Assertions.*;
```

14. Ara ja pots executar el test implementat a la classe `PAC1Task3Test`. Pots executar-ho com hem explicat anteriorment clicant el botó dret sobre la classe i accedint a `Run`, o amb els botons  que veuràs davant de la classe i els mètodes que implementen els tests unitaris.



```

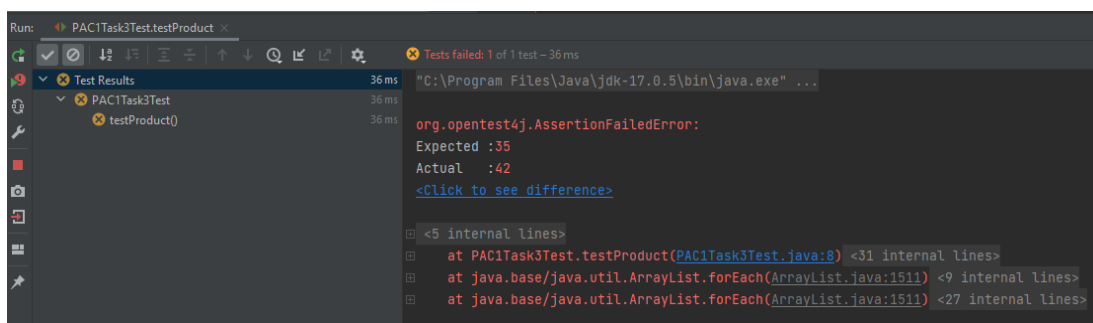
PAC1Task3Test.java x PAC1Task3.java x
AC1Task3
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class PAC1Task3Test {

    @Test
    public void testProduct() {
        assertEquals( expected: 42, PAC1Task3.product( a: 6, b: 7));
    }
}

```

15. Modifica el valor de `expected` (42) per posar qualsevol altre valor que sigui incorrecte, torna a executar i veuràs que l'IDE et mostra els errors del JUnit de la manera: valor esperat i l'actual, així com la traça d'error.



```

Run: PAC1Task3Test.testProduct x
Tests failed: 1 of 1 test - 36 ms
"C:\Program Files\Java\jdk-17.0.5\bin\java.exe" ...

org.opentest4j.AssertionFailedError:
Expected :35
Actual   :42
<Click to see difference>

<5 internal lines>
at PAC1Task3Test.testProduct(PAC1Task3Test.java:8) <31 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <27 internal lines>

```

Exercici 2 (3 punts)

Còpia al teu *workspace* el projecte `PAC1Ex2` que et facilitem amb l'enunciat i obre'l amb Intel·liJ. En aquest exercici codificarem una sèrie de mètodes relacionats amb el càlcul de nombres abundants, deficientes i perfectes.

A continuació has de codificar els `TODO` (de l'anglès, “*fer*”, en espanyol, “*per fer*”, o usant una traducció més lliure: “*pendent*”). Aquest exercici consisteix bàsicament a codificar els 5 mètodes/funcions següents:

- `divsSum`: aquesta funció rep per paràmetre un nombre i retorna la suma dels seus divisors. Per exemple, per a la següent invocació del mètode `divsSum`:

```
divsSum(75)
```

El resultat hauria de ser 124, ja que la suma dels seus divisors (1+3+5+15+25+75) és igual a 124. Aquesta funció torna `-1` si la suma dels divisors (`sum`) és negativa o igual a 0.

(1.5 punts: 1 punt tests proporcionats; 0.5 punts qualitat del codi)

- `isAbundant`: aquesta funció rep per paràmetre un nombre i retorna un booleà indicant si el nombre és o no abundant. Un nombre és abundant quan la suma dels seus divisors és superior al nombre.

(0.2 punts tests proporcionats)

- `isDeficient`: aquesta funció rep per paràmetre un nombre i retorna un booleà indicant si el nombre és o no deficient. Un nombre és deficient quan la suma dels seus divisors és menor al nombre.

(0.2 punts tests proporcionats)

- `isPerfect`: aquesta funció rep per paràmetre un nombre i retorna un booleà indicant si el nombre és o no perfecte. Un nombre és perfecte quan la suma dels seus divisors és igual al nombre.

(0.1 punts tests proporcionats)

- `categorize`: aquest mètode rep un vector amb una sèrie de nombres, calcula quins d'ells són perfectes, deficientes o abundants per a finalment imprimir per pantalla la quantitat de nombres de cadascuna d'aquestes categories ha trobat. Per a la següent invocació:

```
categorize({12, 24, 8128, 0018})
```

Aquest mètode, mostrarà per pantalla el resultat de calcular quins dels nombres son perfectes, abundants i deficientes, seguint el format següent:

```
There are: 1 perfect numbers.
There are: 3 abundant numbers.
There are: 0 deficient numbers.
```

(1 punt: 0.75 punts tests proporcionats; 0.25 punts qualitat del codi)



Requisit mínim per avaluar aquest exercici: el programa ha de passar tots els test proporcionats per `divsSum`, `isAbundant`, `isDeficient` i `isPerfect`.

Exercici 3 (5 punts)

Còpia al teu *workspace* el projecte `PAC1Ex3` que et facilitem amb l'enunciat i obre'l amb IntelliJ. En aquest exercici codificarem diferents mètodes relacionats amb el càlcul de la factura de la electricitat fent servir diferents preus i descomptes en funció d'elements com el consum.

A la classe `PAC1Ex3.java` veuràs definit un vector de set elements, els quals son les taxes fixes associades a cada dia de la setmana (`dailyServiceCharges`). A més, veuràs també una matriu (`unitRatesPerDay`) de tamany `4*7` representant el cost de les unitats per cada dia del mes (per simplicitat, els mesos tenen 28 dies i que durant un mateix dia el cost de les unitats és el mateix).

A continuació has de codificar els `TODO` (de l'anglès, “*to do*”, en català, “*per fer*”, o usant una traducció més lliure: “*pendent*”). Aquest exercici consisteix bàsicament a codificar els 4 mètodes/funcions següents:



El valor retornar per tots els mètodes/funcions que es demanen han de tenir 2 decimals. Per fer-ho utilitza el mètode `twoDecimals` que us donem codificat.

- `calculateTieredPricing`: Aquest mètode calcula, a partir del número d'unitats consumides en un dia concret, el cost per capes/nivells (*tiered*) per aquell dia específic. Per determinar aquest cost, el mètode aplica les següents regles:

```
Les primeres 300 unitats, tenen un cost de 0.04 per unitat.
Les següents 300 unitats (fins 600), tenen un cost de 0.06
per unitat.
Després de les 600 unitats, cada unitat té un cost de 0.014.
```

Aquest mètode ens permet penalitzar als usuaris amb un consum més elevat.

(0.75 punts: 0.5 punts tests proporcionats; 0.25 punts qualitat del codi)

- `calculateDiscount`: Aquest mètode calcula un descompte dependent del número de dies en el que seu consum ha estat per sota de 50 unitats. El descompte es calcula tenint en compte les següents regles (descomptes no acumulatius):

```
Més de 5 dies, 10% del total de la factura.
Més de 10 dies, 20% del total de la factura.
```

Així, per exemple, la següent invocació (11 dies i 32.3 € d'import):

```
calculateDiscount(11, 32.3)
```

La funció retornarà el valor 6.46, el 20% del total (32.3 €).

(0.5 punts tests proporcionats)

- `calculateAdditionalCharges`: aquesta funció calcula, donades les unitats consumides (`units`), els costos addicionals a aplicar al consum fet. Concretament, si el número de unitats consumides és superior a 300, aplica un sobrecost de 0.02 per unitat al total de unitats consumides.

(0.5 punts tests proporcionats)

- `calculateTotalBill`: aquesta funció calcula el cost associat als consums d'un usuari al llarg d'un mes. Com a entrada, espera una matriu de 4 files per 7 columnes amb les unitats consumides per l'usuari per a cada dia del mes. Per a cada dia del mes, el cost de la electricitat associat haurà de ser el que resulta d'aplicar la següent fórmula:

$$\begin{aligned} \text{costDia} = & (\text{unitatsDia} * \text{preuUnitatDia}) + \\ & + \text{costTiered} + \\ & + \text{costosAddicionals} + \\ & + \text{taxaFixaDia} \end{aligned}$$

(3.25 punts: 2.5 punts tests proporcionats; 0.75 punts qualitat del codi)



Consell: Per als càlculs d'aquesta funció és altament recomanable que facis servir les funcions que has implementat anteriorment.



Requisit mínim per avaluar aquest exercici: el programa ha de passar tots els test proporcionats dels mètodes `calculateTieredPricing`, `calculateDiscount` and `calculateAdditionalCharges`.

Format i data de lliurament

Has de lliurar un fitxer *.zip, el nom del qual ha de seguir aquest patró: loginUOC_PAC1.zip. Per exemple: dgarciaso_PAC1.zip. Aquest fitxer comprimit ha d'incloure els elements següents:

- El projecte IntelliJ `PAC1Ex1` completat seguint les peticions i les especificacions de l'Exercici 1.
- El projecte IntelliJ `PAC1Ex2` completat seguint les peticions i les especificacions de l'Exercici 2.
- El projecte IntelliJ `PAC1Ex3` completat seguint les peticions i les especificacions de l'Exercici 3.

El darrer dia per lliurar aquesta PAC és el **16/10/2023** abans de les 23:59. Qualsevol PAC lliurada més tard serà considerada com a no presentada.