Data structures

PR1: Practice 1

Objectives

The objective of PR1 is to ensure that you have the necessary tools to subsequently carry out PR2 (Practice 2) and PR3 (Practice 3).

Additionally, in this practice, we will carry out a basic implementation that will help you review the main aspects of Java programming and object-oriented programming (OOP).

Observations:

- PR1 must be completed individually.
- In the Laboratory forum, questions about installation problems, configuration of your programming environment, or basic programming will be addressed.
- In the classroom (rest of forums), doubts about the conceptual use of ADTs (Abstract Data Types) will be resolved.
- This practice will help you ensure that you have properly configured the necessary work environment for the Data Structures course. As indicated in the teaching plan, the **submission of PR1 accounts for 10% of the final grade for practical activities**.
- The use of artificial intelligence tools is not allowed in this activity. The teaching plan and the UOC website on academic integrity and plagiarism provide information on what constitutes irregular conduct in assessment and the consequences it may entail.

First part (30%)

We ask you to install the IntelliJ[1] Community Edition integrated development environment (IDE) and compile and execute the Java project (DS_PR1). You have TWO options for importing the project:

1. Import the project attached alongside the headlines: DS_PR1.zip.
2. 'Clone'[2] a remote Git repository associated with the PR1 activity:

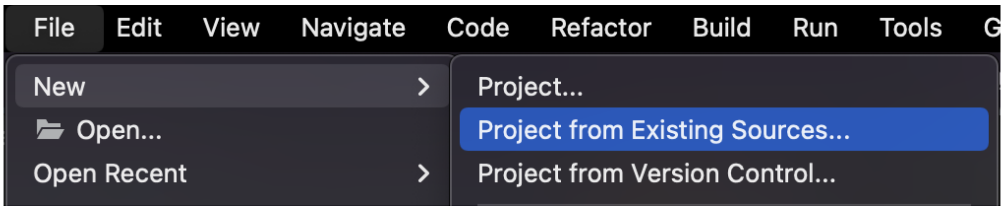
Both options are valid, although it should be noted that Git is currently a reference tool in the industry, and IntelliJ already provides Git support.

Below are the two import options, assuming that IntelliJ IDEA is already installed and that a JDK version 16 or higher is available. Later, the tasks to be performed once the project is imported will be indicated.

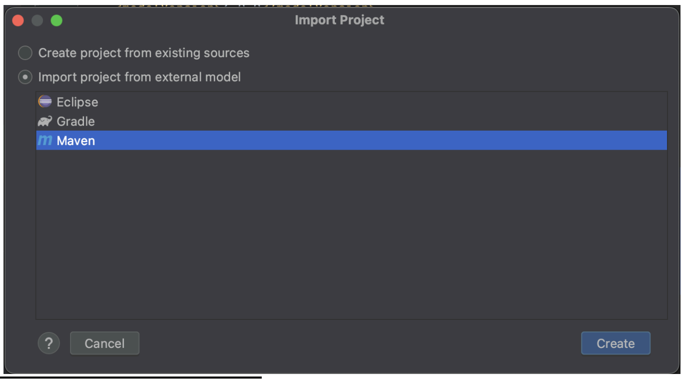
I - Importing a project with existing sources

Steps to follow:

- Unzip the file DS_PR1.zip.
- Once IntelliJ IDE is started, create a new project from existing sources and indicate the directory where the project has been unzipped.

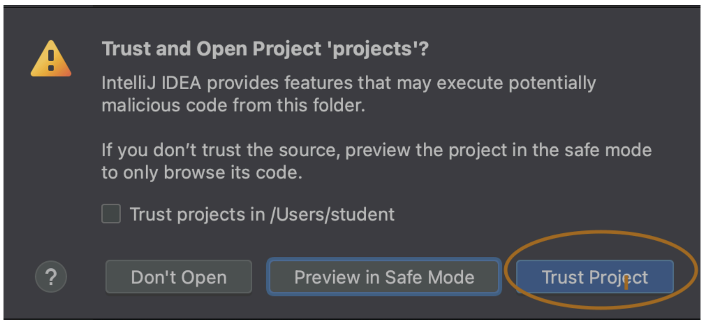


- Indicate that it is a Maven project.


Import Project
Create project from existing sources
Import project from external model
Eclipse
Gradle
Maven
?
Cancel
Create

[1] https://www.jetbrains.com/idea/download
[2] Cloning a repository consists of copying a project from a Git repository to your local machine.

- Trust the project.


Trust and Open Project 'projects'?

IntelliJ IDEA provides features that may execute potentially malicious code from this folder.

If you don't trust the source, preview the project in the safe mode to only browse its code.

☐ Trust projects in /Users/student

If there are no projects initially listed under recent projects, the following screen will appear:

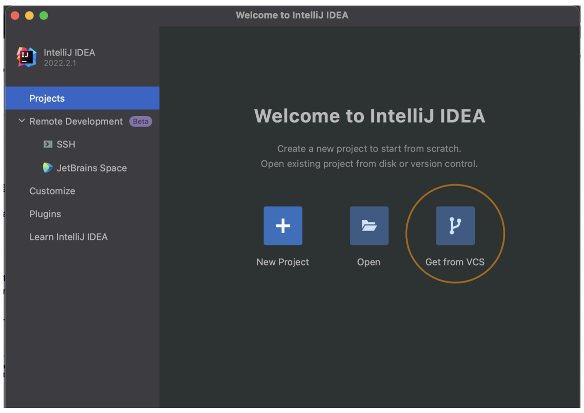
In this case, you must select the “Open” option and follow the steps indicated by the IDE (some have been previously presented).

Once this process is done, the project called DS_PR1 will be deployed.

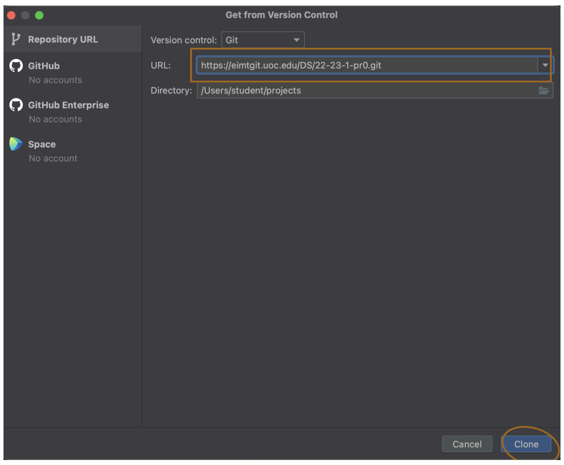
II. Clone the remote repository associated with the PR1 activity

Steps to follow:

1. Select the option to create a new project from a version control system (VCS)

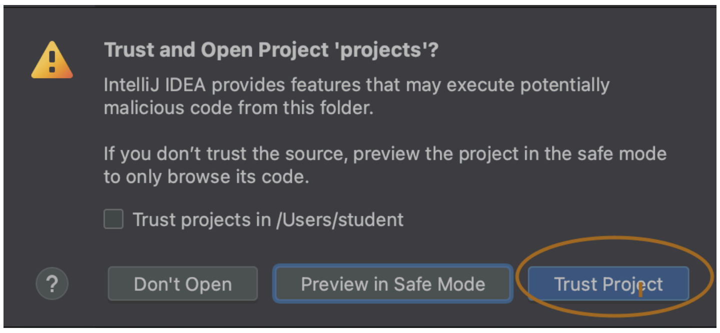


2. Indicate the URL of the GIT repository: https://eimtgit.uoc.edu/DS/DS-PR1.git and 'clone' the repository with the corresponding button.



Version control: Git
URL: https://eimtgit.uoc.edu/DS/22-23-1-pr0.git
Directory: /Users/student/projects

3. Trust the code from the repository developed by the teaching team.



Trust and Open Project 'projects'?

IntelliJ IDEA provides features that may execute potentially malicious code from this folder.

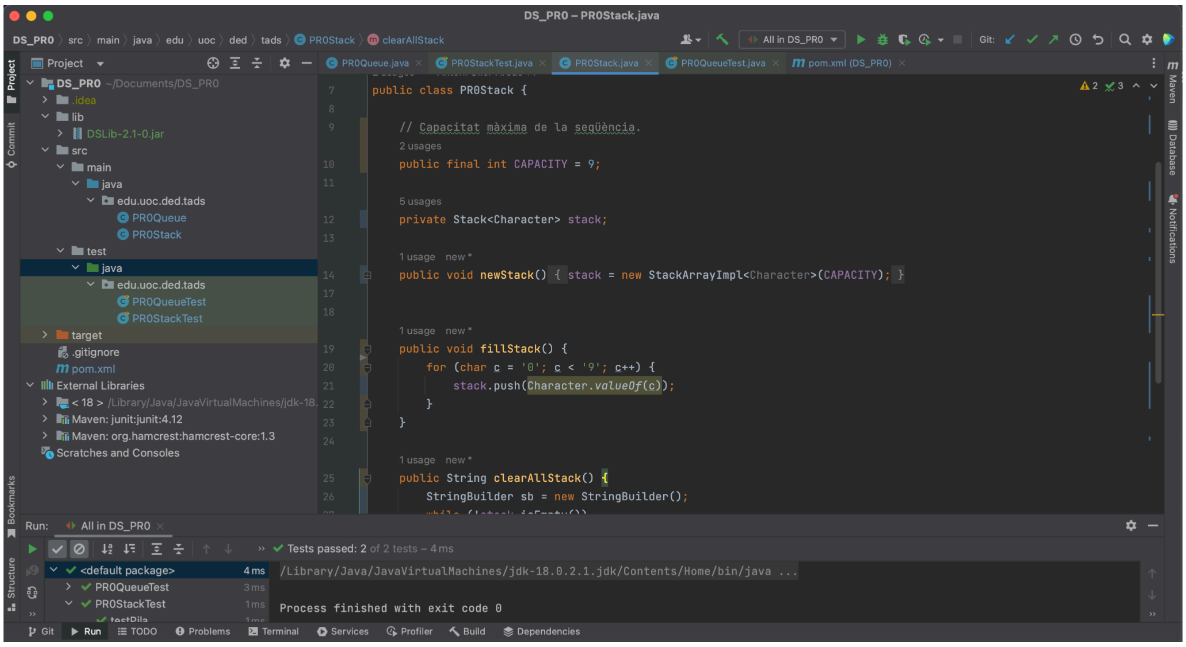
If you don't trust the source, preview the project in the safe mode to only browse its code.

☐ Trust projects in /Users/student

At this point, the project called DS_PR1 will have been deployed and will be in the same state as in the previous section.

III. Tasks to perform once the project associated with PR1 has been successfully imported

Once the project has been imported correctly, the structure of the project can be analyzed.



```java
public class PR0Stack {

    // Capacitat màxima de la seqüència.
    2 usages
    public final int CAPACITY = 9;

    5 usages
    private Stack<Character> stack;

    1 usage   new *
    public void newStack() { stack = new StackArrayImpl<Character>(CAPACITY); }


    1 usage   new *
    public void fillStack() {
        for (char c = '0'; c < '9'; c++) {
            stack.push(Character.valueOf(c));
        }
    }

    1 usage   new *
    public String clearAllStack() {
        StringBuilder sb = new StringBuilder();
```

```
Tests passed: 2 of 2 tests – 4 ms

/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

As you can see in the previous image we have the following directories:

- src/main/java/**: Set of main classes of our project
- src/test/java/**: Set of unit tests of the project
- pom.xml: Maven project definition file. Define the characteristics of the project and its dependencies (libraries)
- lib/DSLib-x.y.z.jar: Library of the course. Implements a set of data structures

The set of unit tests for the project (src/test/*.java files) have been implemented using the JUnit 4 framework. JUnit uses annotations (https://docs.oracle.com/javase/tutorial/java/annotations/) which are metadata that provide additional information about the classes or parts of the classes. Methods with the following JUnit annotations have the corresponding responsibilities:

- @BeforeClass: will always be executed once when the class is initialised.
- @AfterClass: will be executed after all the tests have been executed.
- @Before: will always be executed before each test method.
- @After: will always be executed after each test method.
- @Test: defines a test method.

To complement the knowledge of this framework (JUnit) you can view the following resources: https://www.youtube.com/playlist?list=plk0v_h0fcvpj1dcy4bkwfqrjbdxw7wh7b

Tasks:

- Execute the unit tests that have already been implemented. One option is to use the “Run All Tests” command, which appears when you right-click on the green tests folder.

- Include in the solution document an image of the IntelliJ window, with the ‘project perspective’, showing where you have created, compiled, and executed the JUnit tests for the DS_PRO project, demonstrating that the unit tests have passed successfully.

Deliverable 1: Once the entire import process has been completed, you must attach a screenshot of the execution of the unit tests.

Second part (35%)

Once the previous project (DS_PR1) is created, compiled, and executed, we ask you to modify the code of both the main classes and the test classes of the queue and stack to work with the first 15 values of the following periodic function:

$f(x)=(x \bmod 4)^2$

Example of queue unit test:

```
@org.junit.Test
public void queueTest() {
   assertEquals(this.pr1q.CAPACITY, this.pr1q.getQueue().size());
   Assert.assertEquals(1, pr1q.poll());
   Assert.assertEquals(4, pr1q.poll());
   Assert.assertEquals(9, pr1q.poll());
   Assert.assertEquals(0, pr1q.poll());
   Assert.assertEquals(1, pr1q.poll());
   Assert.assertEquals(4, pr1q.poll());
   Assert.assertEquals(9, pr1q.poll());
   Assert.assertEquals(0, pr1q.poll());
   Assert.assertEquals(1, pr1q.poll());
   Assert.assertEquals(4, pr1q.poll());
   Assert.assertEquals(9, pr1q.poll());
   Assert.assertEquals(0, pr1q.poll());
   Assert.assertEquals(1, pr1q.poll());
   Assert.assertEquals(4, pr1q.poll());
   Assert.assertEquals(9, pr1q.poll());
   assertEquals(0, this.pr1q.getQueue().size());
}
```

Example of a unit test for the stack

```
@org.junit.Test
public void stackTest() {
   assertEquals(this.pr1s.CAPACITY, this.pr1s.getStack().size());
   Assert.assertEquals(9, pr1s.pop());
   Assert.assertEquals(4, pr1s.pop());
   Assert.assertEquals(1, pr1s.pop());
   Assert.assertEquals(0, pr1s.pop());
   Assert.assertEquals(9, pr1s.pop());
   Assert.assertEquals(4, pr1s.pop());
   Assert.assertEquals(1, pr1s.pop());
   Assert.assertEquals(0, pr1s.pop());
   Assert.assertEquals(9, pr1s.pop());
   Assert.assertEquals(4, pr1s.pop());
   Assert.assertEquals(1, pr1s.pop());
   Assert.assertEquals(0, pr1s.pop());
   Assert.assertEquals(9, pr1s.pop());
   Assert.assertEquals(4, pr1s.pop());
   Assert.assertEquals(1, pr1s.pop());
   assertEquals(0, this.pr1s.getStack().size());
}
```

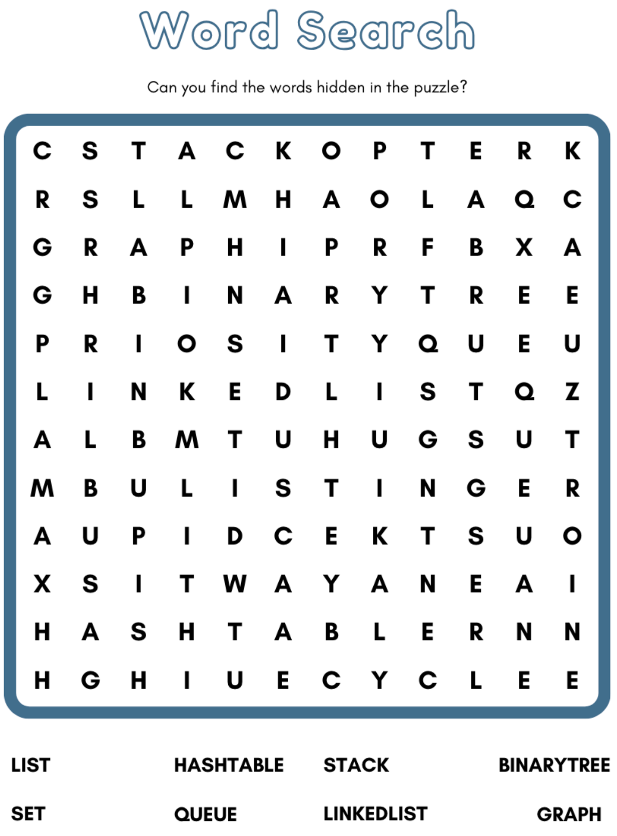
NOTE:

- The data type used in the implementation is characters for both the stack and the queue. Both the queue and the stack must store integers.
- The insertion of values into the data structures must be done in the test class.
- It is recommended to implement the periodic function in an auxiliary class.

Deliverable 2: A compressed file with the modified code.
Optional Deliverable 2: A private GIT repository link with your published project and a screenshot of the web interface showing that repository. You can watch this video explaining how to do this: video link.
The new project must be published in a version control system (GitHub, GitLab, Bitbucket) chosen by the student.

Third part (35%)

The third part of the exercise involves adding a new class **PR1WordSearch** to solve word searches with words related to data structures.



Word Search

Can you find the words hidden in the puzzle?

C	S	T	A	C	K	O	P	T	E	R	K
R	S	L	L	M	H	A	O	L	A	Q	C
G	R	A	P	H	I	P	R	F	B	X	A
G	H	B	I	N	A	R	Y	T	R	E	E
P	R	I	O	S	I	T	Y	Q	U	E	U
L	I	N	K	E	D	L	I	S	T	Q	Z
A	L	B	M	T	U	H	U	G	S	U	T
M	B	U	L	I	S	T	I	N	G	E	R
A	U	P	I	D	C	E	K	T	S	U	O
X	S	I	T	W	A	Y	A	N	E	A	I
H	A	S	H	T	A	B	L	E	R	N	N
H	G	H	I	U	E	C	Y	C	L	E	E

LIST	HASHTABLE	STACK	BINARYTREE
SET	QUEUE	LINKEDLIST	GRAPH

```
@Test
public void wordSearchTest() {
   String in = """
       CSTACKOPTERK
       RSLLMHAOLAQC
       GRAPHIPRFBXA
       GHBINARYTREE
       PRIOSITYQUEU
       LINKEDLISTQZ
       ALBMTUHUGSUT
       MBULISTINGER
       AUPIDCEKTSUO
       XSITWAYANEEI
       HASHTABLERNN
       HGHIUECYCLEE
       """;

   Set<String> words = new SetLinkedListImpl();
   words.add("LIST");
   words.add("HASHTABLE");
   words.add("STACK");
   words.add("BINARYTREE");
   words.add("SET");
   words.add("QUEUE");
   words.add("LINKEDLIST");
   words.add("GRAPH");

   PR1WordSearch wordSearch = new PR1WordSearch(in);
   Stack<PR1WordSearch.Result> out = wordSearch.search(words);

   Iterator<PR1WordSearch.Result> it = out.values();
   PR1WordSearch.Result result = null;

   Assert.assertTrue(it.hasNext());

   PR1WordSearch.Result r = it.next();
   Assert.assertEquals("GRAPH", r.word);
   Assert.assertEquals(2, r.row);
   Assert.assertEquals(0, r.col);
   Assert.assertEquals(PR1WordSearch.Direction.HORIZONTAL, r.direcction);

   r = it.next();
   Assert.assertEquals("LINKEDLIST", r.word);
   Assert.assertEquals(5, r.row);
   Assert.assertEquals(0, r.col);
   Assert.assertEquals(PR1WordSearch.Direction.HORIZONTAL, r.direcction);
```

```
    r = it.next();
    Assert.assertEquals("QUEUE", r.word);
    Assert.assertEquals(5, r.row);
    Assert.assertEquals(10, r.col);
    Assert.assertEquals(PR1WordSearch.Direction.VERTICAL, r.direcction);

    r = it.next();
    Assert.assertEquals("SET", r.word);
    Assert.assertEquals(4, r.row);
    Assert.assertEquals(4, r.col);
    Assert.assertEquals(PR1WordSearch.Direction.VERTICAL, r.direcction);

    r = it.next();
    Assert.assertEquals("BINARYTREE", r.word);
    Assert.assertEquals(3, r.row);
    Assert.assertEquals(2, r.col);
    Assert.assertEquals(PR1WordSearch.Direction.HORIZONTAL, r.direcction);

    r = it.next();
    Assert.assertEquals("STACK", r.word);
    Assert.assertEquals(0, r.row);
    Assert.assertEquals(1, r.col);
    Assert.assertEquals(PR1WordSearch.Direction.HORIZONTAL, r.direcction);

    r = it.next();
    Assert.assertEquals("HASHTABLE", r.word);
    Assert.assertEquals(10, r.row);
    Assert.assertEquals(0, r.col);
    Assert.assertEquals(PR1WordSearch.Direction.HORIZONTAL, r.direcction);

    r = it.next();
    Assert.assertEquals("LIST", r.word);
    Assert.assertEquals(5, r.row);
    Assert.assertEquals(6, r.col);
    Assert.assertEquals(PR1WordSearch.Direction.HORIZONTAL, r.direcction);

    Assert.assertFalse(it.hasNext());
}
```

- **NOTE:**
  To simplify the exercise, the input to this class is a linear sequence of consecutive rows.
- The <u>*search*</u> method of the new class ***PR1WordSearchArray*** receives one parameter: a set (SetLinkedListImpl) of words to search for. The sequence of characters will be passed in the constructor. The result will return a stack with the words found in the original sequence, along with the row and column where the word is located.

- You must code the main PR1WordSearch class and validate its operation with the PR1WordSearchTest class, which you can implement using any test class of the project as a reference, with the following @Test:

Task1: Code the main and test classes indicated above.

Deliverable3: zip file with the project and screenshot of the execution of the unit tests.

Delivery format

The delivery must be made in a single compressed file (ZIP) through the continuous assessment tool that includes:

- A file, in PDF format. This file will contain the solution: the screenshots. required. Please do not copy the statement, put your name on each page (for example, with a footnote), and number the pages.
- A compressed file of your implementation of the second part. Only the sources are requested, you must not include the binary files (*.class) in the compressed file.
- A compressed file of your third party implementation with the same indications as in the previous point.