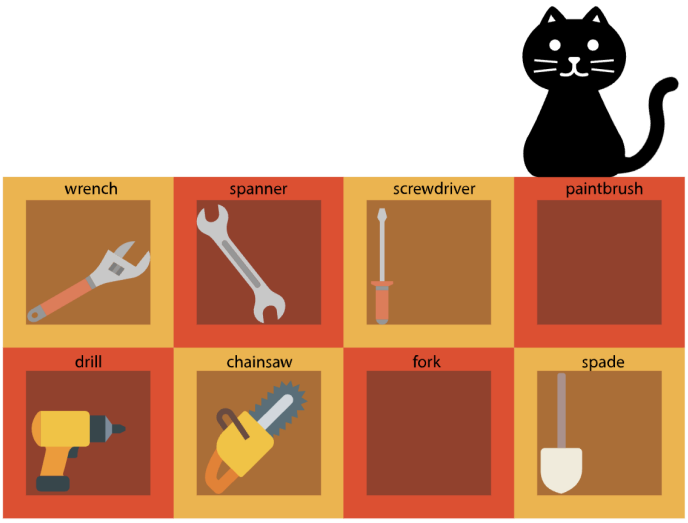

This case study is intended to give you an opportunity on what you think about modular programming and fundamental programming principles, including the processing.

SETUP

Create a new project in your IDE called ToolShelf. Create a Main.java file and copy/paste the program code from the unit page into this file.

Code Analysis Part A

CAPA-1 Complete the following table.

How many global variables are there?	
How many valid tools are there?	
What is the first thing that happens in the game loop?	
What is the last thing that happens in the game loop?	
What happens after exiting the game loop?	
Why does the `validTools` array use the **final** modifier when it is declared?	

Code Analysis Part B

CAPB-1 The `valid` function takes 2 parameters, a String array storing all of the valid tool names and a String representing a tool. Here is the signature[1]:

```
public static boolean valid(String[] vTools, String t)
```

It performs a linear/sequential search, looking for **`t`** in **`vTools`** and returns **true** if it finds t, and **false** otherwise.

Describe the consequence of designing the function as follows:

```
public static boolean valid(String[] vTools, String t) {
        for(int i = 0; i<vTools.length; i++) {
                if(vTools[i].equals(t)) {
                        return true;
                }else {
                        return false;
                }
        }
}
```

[1] A function's signature tells us the name of the function, the return datatype and the datatype(s) of any parameter(s)

CAPB-2 The `borrowTool()` function takes 2 parameters.

Both parameters are references to String arrays.

The first parameter represents all valid tools in the game.

The second parameter represents the current state of the tool shelf.

If a tool is valid and available, it searches the shelf for that tool and replaces it with **null**.

It then **returns** a reference to the updated shelf array.

In the main function, `borrowTool()` is called as follows:

```
currentToolShelf = borrowTool(validTools,
currentToolShelf);
```

How comfortable are you with what is going on here?

Write down any notes or questions you have about this interesting line of code:

CAPB-3 When a tool is borrowed, it is replaced with **null** on the tool shelf. Hence, when processing this array we will sometimes need to check for null. For example, if this is the contents of `currentToolShelf`:

```
{"screwdriver", null, "hammer", "pliers", null, "drill"}
```

And we want to search for a tool:

```
String tool = "pliers";
for(int i = 0; i<currentToolShelf.length; i++){
      if(currentToolShelf[i].equals(tool)){
            System.out.println("Tool found!");
      }
}
```

This code will cause an error when run. As you can see `currentToolSet[1]` is **null**, not a String. So, we cannot use **`.equals()`** with it. We will get a **NullPointerException** error!

Rewrite the above code block to stop the error from happening:

CAPB-4 Describe the purpose of the `emptySpace` function. To help you figure it out, look at its definition and find where it is called in the program:

CAPB-5 The game loop is a `do{}while();` loop. The game could have used a `while(){}` loop or a `for(){}` loop. **Do some research** and reflect on the differences between these 3 loop variations:

Adaptation

After successfully borrowing a tool, if the toolshelf is completely empty a message "NO MORE TOOLS" is displayed. By designing and integrating at least 1 additional function into the program, implement this adaptation.

Testing

During the development of new software, **testing is critical**.
Each **module** needs to be thoroughly tested before it is integrated into the program
The **program** needs to be thoroughly tested as each module is integrated.
When all modules are integrated, the **full program** needs to be thoroughly tested.
Test data needs to be carefully selected during these processes.

The Main program has no syntax errors. It compiles perfectly from Java to **bytecode**[2].
Test the program. There is a **run-time** error in it. Can you find it and fix it?

[2] Computers don't understand Java. Computers only understand 1s and 0s. So, when we compile a program, it is translated from Java to bytecode. Bytecode is a file of 1s and 0s. It is stored in the .class file which humans would find hard to understand!