Practical-Exam-03: Object-Oriented Programming

Due 6 May by 16:55 **Points** 100
Available 6 May at 15:00 - 6 May at 17:00 2 hours

This assignment was locked 6 May at 17:00.

Introduction

This session ends at 16:55

Make sure to complete your submission in time.

Submit to SVN & Websubmission (for every problem solved)

Submission

- Create the repository on the SVN server
- You must checkout only the folder practical-exam-03 from your server
- No other folders from your SVN will be allowed during the practical exam.
- Check if your repository is being track by WebSubmission before start solving the exam.

```
svn mkdir -m "create pracexam3 commit" --parents https://version-control.adelaide.edu.au/svn/<
YOUR-ID>/2022/s1/fcs/week-08/practical-exam-03/tools
```

Assessment

- **This is a practical exam - your work must be entirely your own.**
- Marks for this practical exam contribute 2 marks.
- Marks will be awarded later by your workshop supervisor (30%) and websubmission marker (70%).
- **Due date: 5 minutes before end of this section.**
- **Do Not Forget** To Submit on **WebSubmission (https://myuni.adelaide.edu.au/courses/74996/modules/items/2617266)**
- Submit each part as you go. **Do not leave until the end.**
- **Late penalties:** Only work submitted during your enrolled practical session from a Linux system in the practical lab will be marked.

Regarding functional marks, please consider:

```
(1) only codes that can compile will be marked;
(2) only codes that are in the suggested directory will be marked;
(3) only codes submitted to SVN and WebSubmission before the deadline will be marked;
(4) only codes containing your signature on the top of the file will be marked by tutors;
(5) you will have your markers decreased in 3 points if *.class file present in your folder
s;
```

Signature on your files

Note that all your coding files must contain on the top of it this information:

```
//==================================
// Foundations of Computer Science
// Student: you name
// id: your id
// Semester:
// Year:
// Practical Exam Number:
//===================================
```

Note

- To acquire full marks **(1)** all your functionalities must be working perfectly, **(2)** your code has to be well and proportionally commented, and **(3)** your code must follow correct indentation (4 spaces for every new indentation level) and **(4)** you have to use all the content from latest lectures.
- We argue that you are not just asked to solve a problem but use the more sophisticated way to solve it. For instance, you can solve a problem using ten variables, but it will always be better to solve the same problem with an array.

Practical Exam 03

Classes for your tool collection

In this exam you have to implement code that represents tools that you might have in your collection. Tools have a weight (in grams) and a value (in cents). Because tools have a value they can be considered to be assets. You can use tools and this wears them out a little bit (so their value goes down!).

Some tools are hand-tools (e.g. a hammer and a chisel). Some hand-tools are sharp (e.g. a chisel) and some are not (e.g. a hammer).

Other tools are power tools (e.g. a power drill) and these have a power rating (measured in watts).

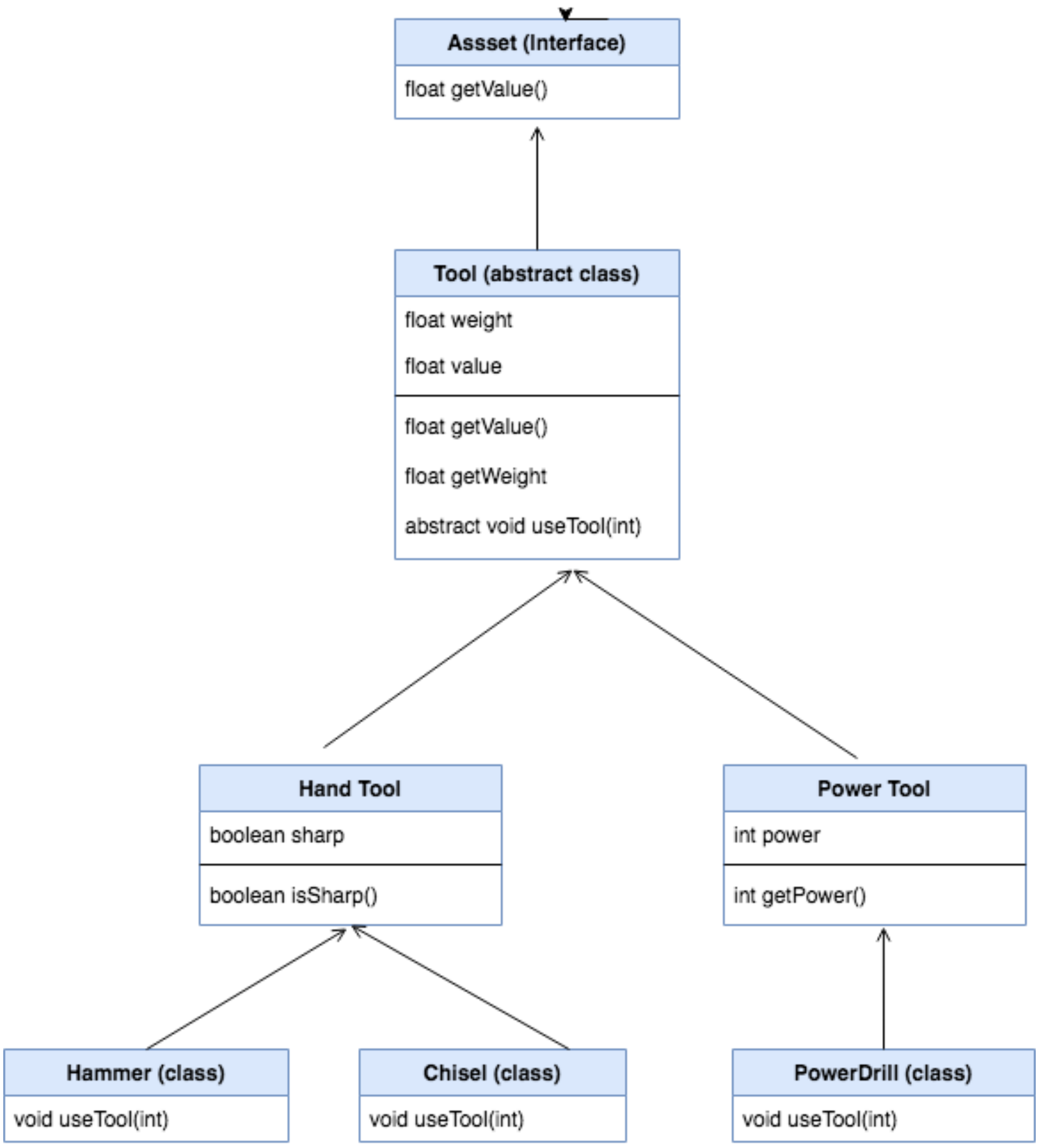
All tools make a sound - which we represent as text that gets printed when we use the tool.

As you might expect you can have multiple tools of the same type (e.g you might have two hammers) and these might have different weight and values.

All of the files you will submit in this exercise must be in **tools** sub-folder that you have created (using the command above) in svn.

Class Diagram

The following diagram shows the classes and interfaces you have to implement in the following questions.



The details for each class will be given in the questions below.

Please make sure you submit your code as you go. Do not submit all of your code for all your questions at the end.

Driver File (for reference only)

To help you test your own code you can use part or all of the following **driver file. (https://myuni.adelaide.edu.au/courses/74996/files/10490615/download?wrap=1)** ⤓ **(https://myuni.adelaide.edu.au/courses/74996/files/10490615/download?download_frd=1)**

Once you have completed your questions then, when you compile and run the ToolDriver driver file above you should get the output shown in **this file (https://myuni.adelaide.edu.au/courses/74996/files/10490613/download?wrap=1)** ⤓ **(https://myuni.adelaide.edu.au/courses/74996/files/10490613/download?download_frd=1)** . Note that you do not have to use this file in your testing. This file is for reference only. It might be helpful as you start to develop your code for the later questions. We will use different tests to the ones shown here.

Problem 01 - An Assets interface

Tools are assets too. In your **tools** folder write Java interface called Asset with a public method signature called getValue(); The return type of getValue() is a float. Submit your code to websubmission to see if if we are able to implement this interface.

Problem 02: - A Tool abstract class

In your tools folder write a Java class definition called Tool.java. This class is **abstract** and should **implement** the Asset interface It should have two member variables:

1. a float value called weight - which stores the weight of the Tool in grams.
2. a float value called value - which stores the current value of the Tool in cents.

You should have the following methods:

1. a constructor which takes two float parameters representing, respectively, the weight and the value of the Tool and initialises the weight and value of this Tool object.
2. a getValue method which takes no parameters and returns the value of this Tool.
3. a getWeight method which takes no parameters and returns the weight of this Tool
4. a void method called **useTool** which takes in an integer parameter representing the number of times the tool is used. It has been decided to defer the implementation of this method to the lowest level classes such as Hammer, Chisel and Power Drill.

Submit your code to websubmission to see if we are able to extend and use this class.

Problem 03: - A HandTool class

In your tools folder write a Java class definition called HandTool.java that extends the Tool class. The designer has decided to <u>defer the implementation</u> of this class to the Hammer and Chisel classes**.**

This class should have one additional member variable called **sharp** that is true if the tool is sharp and false otherwise.

This class has two additional methods:

1. A constructor that takes three parameters: the weight of the tool, the value of the tool and a boolean indicating whether the tool is sharp. The constructor should initialise the relevant variables (hint: t*hink about how you could use the superclass*).
2. A method called isSharp() which takes no parameters and returns the value of **sharp**.

Your class should not yet implement the useTool method. Submit your code to websubmission to see if we are able to extend and use this class.

Problem 04 - A Hammer Class

In your tools folder write a Java class definition called Hammer.java. Refer the class hierarchy diagram when creating this class.

It should have no additional member variables and two methods:

1. A constructor which takes two paraemters: the weight and the current value of the Hammer. The constructor should initialise the **sharp** variable of the superclass with **false** (because hammer's are not sharp). Think how whether you can use the superclass here.
2. The useTool method that takes in an integer representing the number of times the Hammer is used. For each time the hammer is used then 0.01 cents should be subtracted from its value (thus if the parameter is 3 then 0.03 should be deducted from its value). However, the value can never drop below zero. Your useTool method should also print out the word "Bang!" to the terminal. It should only print this word once for each call to this method.

Submit your code to websubmission to see if we are able to use this class.

Problem 05 - A Chisel Class

In your tools folder write a Java class definition called Chisel.java. Refer the class hierarchy diagram when creating this class.

It should have no additional member variables and two methods:

1. A constructor which takes two parameters: the weight and the current value of the Chisel. The constructor should initialise the **sharp** variable of the superclass with **true** (because chisels

are sharp!). Think how whether you can use the superclass here.
2. The useTool method that takes in an integer representing the number of times the Chisel is used. For each time the chisel is used then 0.02 cents should be subtracted from its value. However, the value can never drop below zero. Your useTool method should also print out the word "Scrape" to the terminal. It should only print this word once for each call to this method.

Submit your code to websubmission to see if we are able to use this class.

Problem 06: - A PowerTool class

In your tools folder write a Java class definition called PowerTool.java that **extends** the Tool class. The designer has decided to <u>defer the implementation</u> of this class to the PowerDrill Class.

It should have one additional member variable called **power** that represents the power rating of the tool in watts.

This class has two additional methods:

1. A constructor that takes in a float representing the weight of the tool, a float representing the value of the tool and an int value the power rating of the tool. The constructor should initialise the relevant variables (hint: you might want to call the super constructor).
2. A method called getPower() which takes no parameters and returns the value of the power rating (an integer).

Your class should not yet implement the useTool method (that is why it is still abstract). Submit your code to websubmission to see if we are able to extend and use this class.

Problem 07 - A PowerDrill Class

In your tools folder write a Java class definition called PowerDrill.java. Refer the class hierarchy when creating this class.

It should have no additional member variables and two methods:

1. A constructor which takes two parameters: the weight and the current value of the power drill. The constructor should initialise the power variable of the superclass with **800** (because power drills have a power rating of 800 watts). You might need to call the super-constructor method to do this.
2. A useTool method that takes in an integer representing the number of times the power drill is used. For each time the power drill is used then 0.03 cents should be subtracted from its value (thus if the parameter is 3 then 0.09 should be deducted from its value). However, the value can never drop below zero. Your useTool method should also print out the word "Zssh!" to the terminal. It should only print this word once for each call to this method.

Submit your code to websubmission to see if we are able to use this class.

End of Questions

```
Important
1. all your files must be in your Tools folder.
Your classes MUST NOT contain a public static main function.

2. In order to test your code, please create another class containing Main.java, (don'
t add this to your svn repository) and there you can set your public static main to pe
rform test.

3. Be aware, that your class Main.java might be replaced with a testing class Main.jav
a, to do not add important code in the function Main.java other than testing purposes.
```

Practical Exam 01 (1) (1)

Criteria	Ratings				Pts
Functional	**70 Pts** **Excellent** Your code (1) perform all the functions correctly, (2) use latest concepts learned in class, (3) has a clear, creative and sophisticated way of solving the problem.	**50 Pts** **Good** Your code (1) perform all the functions correctly, (2) use concepts learned in class, (3) has a clear way of solving the problem.	**40 Pts** **Fair** Your code (1) perform almost all the functions correctly, (2) use concepts learned in class, (3) has a way of solving the problem.	**0 Pts** **No marks** You code (1) does not exist.	70 pts
Code Style	**30 Pts** **Excellent** Your code (1) has the right proportion of comments and place line and block comments correctly, (2) follow correct indentation every new indentation level, (3) has good variable naming, (4) has clear organization between tabs and is easy to read.	**18 Pts** **Good** Your code (1) has useful comments and place line and block comments correctly, (2) follow indentation, (3) has good variable naming.	**6 Pts** **Fair** Your code (1) has comments, (2) has variables, (4) has clear organization.	**0 Pts** **No marks** You code (1) does not exist.	30 pts
					Total points: 100