

Informatics 1: Object Oriented Programming

Tutorial 03

Week 4: Code Golf

Brian Mitchell (brian.x.mitchell@ed.ac.uk)

Vidminas Vizgirda (s1750767@ed.ac.uk)

1 Introduction

In this tutorial, we will practice skills that a good programmer or Computer Scientist needs frequently. These include understanding problem scenarios, designing programs, and understanding the relationships between code and design. This is partly why there is a reflective task at the end: do not skip this. This tutorial is also about seeing how programs are built from small pieces.

This tutorial gives you practice for Assignments 2 and 3, though the tasks deliberately differ from the Assignments' contents. This tutorial includes practical skills such as using markdown text formatting and using zip files for IntelliJ projects.

2 ASCII Dice

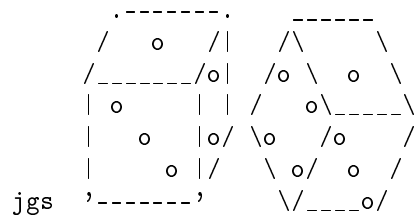


Figure 1: ASCII Art Dice by [Joan G. Stark](#). Retrieved from the [ASCII Art Archive](#). Copyright © 1996-01 Joan G. Stark. You can learn more about Stark on [Wikipedia](#).

In this tutorial you will work on designing or coding or analysing a program to “pretty print” the faces of a standard 6-sided die (plural “dice”, singular “die”). You can imagine this as part of a larger project such as a digital board game. You will also look at how the design and code are explained to beginners — this aspect is directly relevant to the Inf1B assignments.

“Pretty printing” means displaying text in a fancy or elaborate way. For this task, our goal will be to take a digit from 1 to 6 (the roll of a dice) and print a picture using plain text, a technique called [ASCII art](#). The output should resemble the dots on the appropriate face of the die.

We won't be drawing complex art like the above illustration. Instead, it will be enough to draw the top-down view of a rolled die face, using spaces, O characters, dashes, and pipe symbols, like below:

```
1:      2:      3:      4:      5:      6:
-----
|  |  |  |0|  |  |0|  |  |0 0|  |0 0|  |0 0|
| 0|  |  |  |  |  |0|  |  |  |  |0|  |0 0|
|  |  |  |0|  |  |0|  |0 0|  |0 0|  |0 0|
-----
```

Task 1 - Team up

◀ Task

Pair up with a partner to work with during this tutorial. If you have an odd number of people, you can also work in a group of 3.

Like in previous tutorials, you should use pair programming, switching between driver and navigator roles. The tutors will no longer prompt you when to switch, so you should decide when you are comfortable to do this yourselves, but make sure that everyone in your pair (or group of 3) gets a chance to spend some time in both roles.

Task 2 - Download and open the project design and code template

◀ Task

Download the `ASCIIDice.zip` archive from the course materials site (where you found these tutorial instructions).

Instructions for opening the project and navigating the menus:

1. in Explorer / Finder / Nautilus / other file manager: unzip the downloaded zip file (for Mac users, this might already be automatically done for you when you download the file)
2. optionally move the unzipped folder to somewhere more convenient for working
3. in IntelliJ: Open a new project, navigate to the directory containing the `ASCIIDice.iml` file BUT open the directory NOT the `.iml` file: this means choosing the directory in the dialogue box not any of the files inside it: it means clicking the directory name and then clicking OK
4. IntelliJ will now open the project but the IntelliJ window doesn't show you much initially, which can make you think nothing has happened, so:
5. open the Project view (`Alt+1` or click on the "Project" tab in the top-left of the screen with sideways writing)
6. open the Structure view (`Alt+7` or click on the "Structure" tab in the bottom-left of the screen with sideways writing)
7. in the Project view double click on `readme.md` to open the project description file
8. in the Project view, expand the `src` folder to see the provided source code. You should see one file: `PrintDice.java`
9. At the bottom of the IntelliJ screen, click on the "TODO" tab. It will show you all comments in the project code starting with "TODO". You can click on items in the "TODO" tab to navigate to them in source code.

Task 3 - Read the introduction to Markdown

◀ Task

Markdown is a widely used standard for marking up text files in a simple way. Although `markdown` only uses basic text characters it allows font changes, colour, automatically formatted code listings, automatically formatted tables, footnotes, hyperlinks, and graphics (embedded via a hyperlink). Magic! You can use `markdown` on Piazza which will give you regular practice: the fastest way to master something's usage. There are different styles ("flavours") of `markdown` but as long as your `markdown` works in IntelliJ's viewer and is consistent, that is acceptable. You need to use `markdown` for Assignments 2 and 3 but this is just an introductory glance.

In the project, open the `markdown.md` file. Take about 5 minutes to read its contents. The file has instructions for how to view `markdown` in editor and preview modes – try out the available options.

Complete the IntelliJ configuration steps.

Skip the further reading links for now – you can come back to these after the tutorial, if you want.

Task 4 - Read the project README file

◀ Task

The `readme.md` file walks through the setup of this project, some hints, and steps to implement an ASCII Dice printing program.

Read these instructions and clarify any confusing parts. You might want to draw on paper or otherwise visualise how this program will work.

If you're unsure about something and none of the partners in your group know, try finding the answer on the internet, and if you still can't find it, then ask a tutor for help.

You might need around 10-15 minutes for this.

Task 5 - Implement and test code

◀ Task

Now that you have the design in mind, write code to implement it. You should populate the provided `PrintDice.java` file and add additional files (if needed).

If you feel confident in writing the code, you can make improvements the proposed design, but this is optional.

Take your time in writing code. If you get stuck, don't hesitate to ask for help. If you finish quickly, move on to the next tasks, but if you need more time – that's okay, just leave at least 20 minutes for the writing, reflection, and exporting tasks before the end of the tutorial, but you can skip the rest.

Task 6 - Write up the program design in your own words

◀ Task

Open the `walkthrough.md` file. It contains a design write-up template.

Take at least 10 minutes to fill in this template, describing, in your own words, how your implementation works.

Task 7 - Write reflections

◀ Task

Reflecting on each tutorial (and lab sheet) is important to strengthen your experience and skills. It helps reinforce your learning and can reduce the amount of revision needed in future. Do not underestimate the value of reflective thinking. It is time invested. It helps you remember situations and solutions for future use, thus saving you time and making you more effective. It also helps you consciously make connections between different aspects of programming, software development, and computer science generally, such as how programming languages reflect aspects of hardware, or how good interface design reduces errors.

Your notes can be about anything you feel is important but should probably include:

- any unexpected difficulties;
- any difficulties you expected that did not occur;
- any useful knowledge you gained;
- any existing knowledge you reused;
- any connections you have made.

Open the `reflections.md` file and add your reflections there.


Take at least 10 minutes to answer the questions and prompts in the provided template. Also try to think how each item might be useful to you in future, even if you do not yet understand it fully (or at all).

Task 8 - Export your work

◀ Task

For Assignments 2 and 3 you need to submit an IntelliJ project that has been properly exported as a zip file. This is one reason why we are insisting on using IntelliJ and why you have been practicing with IntelliJ.

1. `Build >> Rebuild Project` and fix any compilation errors — in assignments, code which does not compile (for any reason, no matter how trivial) scores 0.
2. Make no more changes — if you do return to step 1.
3. `File >> Export >> Project to Zip file...` and rename the resulting zip file according to the instructions in the `readme.md` file you were working with.
4. Ensure you do not include a previous exported zip file in the latest project export (a zip inside a zip) because this means if an autotester is used on your code it is not guaranteed to run the latest version of your code.
5. Copy your zip file somewhere else, unzip it, and check its contents are what you expected.

 That's it for the main part of the tutorial!

All the steps you practiced here are very similar to what you will be required to do for Assignment 2.

You can get a bit more practice with the optional tutorial tasks that follow. If you don't get to them during the tutorial, you can also complete them in your own free time. Feel free to ask for support on Piazza if you get stuck.

 Sample solutions for this task are provided on the [Inf1B course materials page](#).

3 Optional: Code Golf and microwave timers

The first task was inspired by a “Code Golf” challenge: [Convert numbers to dice patterns](#).

Task 1 - Another challenge

◀ Task

See this CodeGolf challenge page: [Doing a little trolling with the microwave timer](#).

It will be our next task. Create a new IntelliJ project for this task.

Task 2 - Implement a solution

◀ Task

For fun, you could try re-implementing some of the proposed solutions in Java – some are usable. However, none of them are readable.

The real task is to implement a solution that works and is also easy to understand.

Task 3 - Describe the design

◀ Task



Figure 2: “Golf Course” by pianoBrad is available in the [Public Domain](#).

Code Golf is a website of coding challenges, where anyone can post challenges and solutions. Its name mirrors the game of golf where you win with the lowest number of strokes, except in code golf, you win by writing a solution with the lowest number of characters in your source code. Shortening of source code, often to extremes, is also sometimes called minifying or uglifying.

However we are not playing Code Golf with our work. Though we do want you to eventually be able to write *elegantly* concise code, in Inf1B the most important things are writing code that is readable and easy-to-understand – even if this means the code is not the shortest solution possible. Well designed code is highly prized in both academic and corporate programming contexts.

Copy the `walkthrough.md` file from the previous project and write up the design of your new implementation.

Task 4 - Reflect


◀ Task

Copy the `reflections.md` file from the previous project and add any new reflections you have after completing the above tasks.

Task 5 - Export your work

◀ Task

Follow instructions from the previous task to export your work to a zip archive.

 There are no sample solutions for this challenge. There are many different ways to get it right. If you would like feedback on a particular aspect, please post a question on Piazza.