Coding Conventions

When working on a large project, especially when the code is shared by many people, it is important to conform to a unified and consistent coding style.

In this course, we require that you conform to the coding conventions suggested by Sun (http://www.oracle.com/technetwork/java/codeconventions-150003.pdf). All the code that you submit must conform to this coding style.

Note that some of the code we present in class, in the recitation session (tirgul) or in the assignment descriptions, do not follow these conventions. This is because our primary concern for this code is to be short, concise, and to fit on a small space. However, whenever we distribute files, they will follow the conventions.

Automatic code style checking with Checkstyle

In order to verify that your code follows the coding style, you can use the `checkstyle` software, with the configuration (data/checkstyle/biuoop.xml) we provide. You can learn more about the checkstyle project here (http://checkstyle.sourceforge.net/).

We will run automatic checktyle on all the code you submit. If the output of checkstyle indicate any deviation from the coding convention, we will reduce 10 points of your exercise grade.

Note that the checkstyle test is a necessity, but not a sufficiency - your code should also be well documented with comments as appropriate for being a readable code. In case your code will be manually inspected and the graders will face critical coding style problems, such as under-documented or ill-styled code (e.g. meaningless names or extremely long functions), we may reduce the grade more significantly and even give a grade of 0.

Please make sure to fix such errors *BEFORE* submitting the code, by verifying that you pass the checkstyle test with NO errors.

Using Checkstyle

Download the files:

- checkstyle-8.44-all.jar (data/checkstyle/checkstyle-8.44-all.jar)
- biuoop.xml (data/checkstyle/biuoop.xml)

The first file is the checkstyle software, the second is the configuration file we provide, specifying the coding convention we require. You can have a look at the content of `biuoop.xml` if you are interested.

Running checkstyle:

```
> java -jar checkstyle-8.44-all.jar -c biuoop.xml JavaFile1.java JavaFile2.java ...
```

The `...` at the end indicate that you can add any number of java files. You can also specify many files using wildcards:

```
> java -jar checkstyle-8.44-all.jar -c biuoop.xml path/to/code/*.java
```

A small exercise

This small exercise will help you make sure you get checkstyle to work:

- Download `checkstyle-8.44-all.jar` and `biuoop.xml` from above.
- Download the following code file FailingCheckstyleCode.java (data/checkstyle/FailingCheckstyleCode.java)

Run the following command

```
java -jar checkstyle-8.44-all.jar -c biuoop.xml FailingCheckstyleCode.java
```

You should see many checkstyle errors.

Fix the code so that checkstyle does not produce any more errors.

Checkstyle in IntelliJ

From Assignment 2, you'll find it useful to write your code in an IDE such as IntelliJ, Eclipse, VSCode, or any other IDE that you like. In some IDEs, you can install a plugin for integrating checkstyle though it's not mandatory. The following section explains how install it in IntelliJ.

Below the following steps to install a plugin (https://plugins.jetbrains.com/plugin/1065-checkstyle-idea) for integrating checkstyle in IntelliJ.

1. Go to Settings (Ctrl + Alt + S) -> Plugins -> search for "checkstyle", install "CheckStyle-IDEA" and reload IntelliJ.
2. Go to Settings -> Editor -> Inspections -> make sure you activate the Checkstyle inspection
3. To configure the checkstyle with our configuration file (biuoop.xml): Settings -> Other Settings -> Checkstyle. There:
    - In Checkstyle version, select 8.44.
    - Under Configuration File, click Add (+). Provide a name (Description) and then Browse to the biuoop.xml configuration file we have provided. Click Next and Finish. Then, back in the Settings window, click Apply.

Now, you have a Checkstyle panel in the bottom pane. Click it. You can run a check on a certain file or all on all files in the project. You will see the problems also as code inspection inside the main window, similar to error corrections in Word. Hover the mouse on the highlighted code fragment to see the checkstyle error message.

Installing Apache Ant

Apache Ant is a software, thus it requires installation wherever you want to use it. We will install the 1.10.13 version.

Pre-requisite

It is assumed that you have already downloaded and installed Java Development Kit (JDK) on your computer.

Download

For Mac users who use brew to install packages (highly recommended in general), you can simply install ant using `brew install ant` and move to Verify Installation.

For Windows and Linux, proceed as follows:

- Download Apache-Ant-1.10.13 binaries (https://dlcdn.apache.org//ant/binaries/apache-ant-1.10.13-bin.zip). It is a zip file.
- Unzip the zip file to a convenient location (e.g. `C:\Users\User`). To unzip on Windows, you can use 7-zip (https://www.7-zip.org/) or similar programs.

Set environment variables (Windows and Linux)

- Create a new environment variable called ANT_HOME that points to the Ant installation folder, in our example, the `C:\User\User\apache-ant-1.10.13-bin` folder.
- Append the path to the Apache Ant batch file to the PATH environment variable. In our example this would be the `C:\User\User\apache-ant-1.10.13-bin\bin` folder. You can now run `ant` commands from anywhere on your system.

If you are not sure how to create or modify an environment variable, here are simple guides for Windows (https://www.architectryan.com/2018/08/31/how-to-change-environment-variables-on-windows-10/), Linux (https://www.serverlab.ca/tutorials/linux/administration-linux/how-to-set-environment-variables-in-linux/), and Mac (https://medium.com/@youngstone89/setting-up-environment-variables-in-mac-os-28e5941c771c) (note you should make it a persistent setting).

Verify Installation

To verify the successful installation of Apache Ant on your computer, type `ant -version` on your command prompt.

You should see an output similar to −

```
Apache Ant(TM) version 1.10.13 compiled on January 4 2023
```

Integration with IDEs

Common Java IDEs, such as IntelliJ and Eclipse, support the usage of Ant. You can copy the `build.xml` (that we will provide you for each assignment) into your project directory and then edit your build configuration so that it uses Ant targets specified there.

For integration with IntelliJ, see Ayal Klein's video (https://www.youtube.com/watch?v=WNHGXAygwZw&feature=youtu.be) in hebrew.

These instructions are required for ALL the code you submit

Please follow them completely, or we will not be able to grade you code, and you will receive a 0 grade for the assignment.

- Use the submit (https://submit.cs.biu.ac.il/) system.
- All your code should reside in a single `.zip` file named `assN.zip` where `N` is the assignment number.
- When the zip file is extracted, it should create a sub-directory called `src` and the ANT config file `build.xml` (see next section). All your other code should reside under the `src` directory. You can create other directories if you want. The command line to correctly zip the assignment is `zip -r assN.zip build.xml src`.
- When compiling your code, we will export the compiled `.class` files into a sub-directory named `bin`. This is a common project structure.
- We may ask you to call certain classes in certain names. Please do so.

Checkstyle

- You must follow the [[CodingStyle]] and pass the checkstyle test with no errors.

User IDs

- On the main class `.java` file (or any file with a `main` function), add your ID in a comment on the head of the file.

ANT - A New Build Tool

In your "Introduction to Computing" course, you have learned about a simple utility named `makefile`. You have used a makefile to organize your compilation neatly, so that instead of typing a complex or cumbersome command every time you want to rebuild your project, you could just type `make compile` and the makefile knew what to do.

You should now understand that `makefile` is just one instance (in fact, a quite primitive one) of a class of software programs termed **build tools**. Build tools are programs that automate the creation of executable applications from source code. They are aiding software development by automating a wide variety of tasks that software developers do in their day-to-day activities, such as:

- Managing and downloading dependencies.
- Compiling source code into binary code.
- Packaging that binary code.
- Running tests.
- Deployment to production systems.

Throughout the assignments of this course, we will use Apache Ant (https://ant.apache.org/) as our build tool. Ant is a simple build tool; Nowadays, real-world Java projects use more powerful build tools such as Maven, SBT or Mill. However, it fits our purpose, and it does provide important utilities that will aid us to unify the testing of your assignment across platforms.

In Ant, the main configuration file is called `build.xml`, which is somewhat equivalent to the `makefile` you are already familiar with. As you can understand from its extension, it is an XML (https://en.wikipedia.org/wiki/XML) file, which specify a certain format for files so that machines can unambiguously retrieve information from it.
Similarly to `makefile`, the `build.xml` file specifies certain *targets* (i.e. automated tasks you'll want to use over and over again) and how to perform them. After writing a `build.xml` and putting it in your current directory, you can execute a target by running the command `ant <target-name>`. This will invoke the `Ant` software, which in turn will search for and read the `build.xml` and perform the the specified task. For more information about Ant, refer to online tutorials such as tutorialspoint (https://www.tutorialspoint.com/ant/index.htm).

Fortunately, we provide you in each assignment with a suitable build.xml file that will fit all your needs. Your requirements are:

1. [[Installing Ant]] on your personal computer (where you develop your code).
2. Download our provided `build.xml` for each assignment and put it under the root directory.
3. Before you submit, test your assignment by compiling and running it **using Ant**. It's crucial that you do so (and don't rely on the IDE `run` button for example) because our graders will use ant to compile and run you code - so this is your test.

To compile all your source files (which as mentioned, should reside in the `src` subdirectory), run: `ant compile`. To execute a program, use `ant run` when there is only one main class. In some of the assignments you have multiple tasks for which we instruct you to provide more than one main class (that is, a class with a `main` function you can run); in that case execute each task with `ant runI` (`I` being the task number).

In the `build.xml` we will provide for each assignment, you will also have:

- a `clean` target for cleaning up the binaries you have compiled. Please use it before you zip your assignment's root directory - you should not submit any `.class` files.
- a `check` target - this is for your convenience.

For Windows and Linux only: if you copy our provided `checkstyle-8.44-all.jar` and `biuoop.xml` to your root directory, `ant check` will run the checkstyle on all the `.java` files under `src` (and recursively its subdirectories, if any). For Mac users, please run the checkstyle using the command line, as described in [[CodingStyle]]

You should download the `build.xml` file we will provide in the assignment, and include it without changes in the zip directory you submit. This is mainly for the convenience of grading your assignments. You are required to verify your assignment is properly compiling and running using ant, since this is how graders will check your work.

What we will run on your submission

We should be able to run the following commands (on unix) to run your code:

```
unzip ass1.zip
ant compile
ant run
```

To execute a `run` target with command line arguments (e.g. `10`, `arg2`, `3`), run:

```
ant -Dargs="10 arg2 3″ run
```

An example

Assuming we had assignment number 99, with two task: in task 1 your program was requested to print "Hello Task1", and in task 2 your program was requested to print "Hello Task2". Then the following file is a valid submission: ass99.zip (data/ass99.zip)