Assignment: Grade

This assignment has zero weight, i.e. it won't count towards your mark for the unit, but don't miss it out. The aims are:

- to start using Java
- to try the automatic marking system
- to sort out problems with the lab MVB 2.11 (ask in MVB 3.41)
- to sort out installation of Java on your own computer

The deadline is Monday 29th January. It is possible to submit up to a week after that, with mark penalties. The mark penalties won't matter because the assignment doesn't count, **except** that it's a strong indication that you are not keeping up.

Before you start, you might like to work through the early parts of the Getting Started aside, to install Java on your own laptop, or check some of the details of how to use it. You might also like to work through the Java 1 aside, to check some of the details of Java for simple programs.

The Program

The idea is to complete this program:

Grade.java

It takes a percentage mark that you might get for a coursework, exam, unit or degree, and turns it into a grade:

```
> javac Grade.java
> java Grade 70
Distinction
```

It is not meant to cover anything besides MSc marks, nor the various borderline and special case issues that arise in real life. You will want to run the program like this most of the time:

```
> java Grade
```

The program then runs all the tests and prints out an error message for the first test that fails.

Your task is to get all the tests to pass. What is the detailed specification of what the program should do? The tests and their comments form the specification. Just work though them. What happens if a test fails? It crashes the program with an error message and a backtrace containing the line number of the test, so you can investigate and debug as normal.

You should find the `grade` tests easy enough to pass. After that, you can start work on the `convert`function. The `convert` function can be implemented by dealing with each possible length of the input string in turn (with three characters as the maximum valid length). There are also cleverer and more general ways, involving exceptions for example.

Translations

Your first task is to alter the `grade` function near the top. You don't need to touch anything else except, later, the `convert` function. Inside functions, Java programs are quite like C - they use semicolons, curly brackets, `if` statements and so on, which have the same syntax as C. Here are a few things that you may need to know how to do in Java, with their C translations:

```
String s = "abc";                 const char *s = "abc";
```

Strings are of type `String`, not `char *`, and they are immutable. In other words you can't change them, other than by constructing a new string.

```
int m = Integer.parseInt(s);      int m = atoi(s);
```

The function `Integer.parseInt` converts from a valid string to an integer.

```
int n = s.length();               int n = strlen(s);
```

The function `length`, attached to every string `s`, finds its length.

```
char c = s.charAt(i);             char c = s[i];
```

The function `charAt`, attached to strings, extracts the `i`'th character, instead of indexing, because a string is an opaque object. (In Java, characters are two bytes long.)

```
if (s1.equals(s2)) ...            if (strcmp(s1,s2) == 0) ...
```

String comparison uses the `equals` function.

```
    if (c >= '0') ...                 if (c >= '0') ...
```

No difference.

```
    System.out.println(s);            printf("%s\n", s);
    System.out.println("n=" + n);     printf("n=%d\n", n);
```

Use this for debugging etc. If you think it is verbose, define an abbreviating function.

Submission

I am not planning to read your program or give you any feedback (unless anything goes wrong), and your mark won't count towards the assessment for the unit. The purpose of submitting is (a) so we and you know that you are keeping up with the unit and (b) for you to build up a portfolio on SAFE of the Java programs you have written. If you want feedback, ask in the lab (while you are doing the work, if you want), or find me in my office.

The marking of your program will be automated. Your `grade` and `convert` functions will be called and tested from a separate program. For this to work, here are the rules you must follow:

- just submit `Grade.java`, nothing else (your program will be compiled during marking)
- don't change the name (e.g. `grade.java` will get a mark of 0, or a just a mark reduction if I'm in a good mood)
- don't change the arguments or result type of the `grade` or `convert` functions
- don't make the `grade` or `convert` functions private
- remove or comment out or switch off any debugging print statements inside `grade` or `convert`
- make sure your program compiles and runs (bar test failures) before you submit it
- beware of doing a last minute 'fix' which breaks the program

The exact way that your program will be marked is that my marking program will be compiled alongside your program. It will call your `grade`and `convert` functions, but run its own tests. My tests will be the same as yours, except (a) using my copy guards against the possibility that you might have changed your version of the tests and (b) a failure doesn't crash my program, but instead my program counts the tests which pass before one fails.

The marking is incredibly simple: there is 4% for each of the 25 tests which passes, up until one fails. I expect most of you to get 100%, but don't expect to be able to get such high marks later!

I like to put deadlines at midnight on the Monday after the week or weeks when you are meant to be doing the work. The weekend is in case you haven't finished when you should, and the Monday is in case you had trouble over the weekend and need help. Some people think midnight gives the wrong message, suggesting that students should work all hours and not have a life. I intend it to give the message that students shouldn't be working right up to a deadline at all. Besides, there is no logical, sensible daytime deadline that everyone can agree on.

Extra exercises

If you finish and want to try more small programs, here are some ideas.

The Chart program Chart.java in the Objects chapter (which uses Counter.java) is monolithic. It is essentially all in one large, rigid piece, the `run`method. Rewrite it in the 'clean code' style, with tiny self-describing methods.

- ▶ Hint 1
- ▶ Hint 2
- ▶ Answer

In the Components chapter, the Clock program consists of the classes Clock.java Counter.java. Make the clock optionally 12-hour instead of 24-hour. Give the hours counter a limit of 12, and add an a/m, p/m counter with a limit of 2. For the hours, the clock should produce `12:xx` instead of `00:xx` (which is a weirdness of 12-hour times). For a/m, p/m, the time should be displayed with a suffix of `am` or `pm`. (These changes are a bit awkward, because different counters behave differently, yet you want to stick with a single, general Counter class, but do the best you can.)