Universiteit van Amsterdam

Programmeertalen

Schrijfopdracht

L. Thomas van Binsbergen

Case study: Java

Deadline: Maandag, Week 8

Practical information

Before answering questions: read through the whole assignment and discuss the resources (below) within your group.

Make sure you include your names, student numbers and group number in your submission. This assignment is executed in a group of three. Only submit one solution per group. Groups are self-registered on Canvas. You can write your answers in any medium you like (text editor, markdown editor, LaTeX/Overleaf, Word, etc.) as long as you **submit a pdf document**. If in your answers you wish to include code fragments, then working in LaTeX, markdown, or plain text are recommended. Write specific and concise answers and include references when citing or quoting outside material.

The questions with point indicators sum up to 80 marks out of a maximum of a 100. Good answers to at least *some* (certainly not all) 'bonus points' questions are therefore required to score above an 8.0 for this assignment. Most questions require multiple answers and/or an accurate explanation, making it easy to miss points. Therefore bonus points should also be considered for making up points missed elsewhere. This assignment plays a *formative* role, meaning that it serves as a tool for learning rather than examination. The formative role is reflected in the low relative weight of this assignment. The primary objectives are for you to learn to study programming language documentation, study programming concepts and make comparisons with other languages.

Introduction

The Java programming language is one of the most popular languages for object-oriented programming and is taught as a first language at many universities (including the UvA until 2021). Java is sometimes referred to as a purely object-oriented language because almost everything in Java is an object, including arrays. The only exceptions are the so-called primitive types. Moreover, the language is designed with object-orientation in mind. Originally Java was used mostly for developing web applications in the form of *applets*. However, it is now one of the most used general-purpose programming languages for developing all sorts of applications. Compared to, in particular, C++, the Java design includes a number of design decisions that limit the freedom of the programmer, removing also the freedom to make certain mistakes (e.g. no pointer arithmetic). In this assignment you will investigate some of these design decisions.

The main resource is the official Java language documentation found at the link given below. However, feel free to use your own resources (for example, W3schools), but be aware of language versions: Java has undergone significant extensions over the years.

- The documentation of Java version 17:
  `https://docs.oracle.com/javase/specs/jls/se17/html/index.html`
- W3schools Java tutorial:
  `https://www.w3schools.com/java/default.asp`

Primitive types and reference types

Almost every value in Java is an object, except the values of primitive types. The distinction between primitive types and reference (or object) types is relevant to both syntactic, semantic and pragmatic aspects of the language.

Q1 (5 points) For every primitive type there is a reference type counterpart. Name at least 3 primitive types and their reference type counterpart.

Q2 (5 points) The *boxing conversion* converts the value of a primitive type to an object of the corresponding reference type. Give a code example on which the Java compiler performs this conversion automatically and explain why this conversion is helpful to the programmer in this example.

Q3 (10 points) Given that primitive types have a reference type counterpart, one might argue that the primitive types are redundant. Provide at least one argument motivating the existence of primitive types in Java and at least one argument against their existence.

Q4 (10 points) Which parameter-passing strategy does Java apply? Explain your answer.

Q5 (5 points) What is the difference between the following two valid Java expressions in which `s1` and `s2` are both of type `String`.

```
s1 == s2
```

```
s1.equals(s2)
```

Q6 (5 points) In your own words, explain the difference between `==` and `.equals()` for all types.

Q7 (0 points) Reflect on the previous questions and discuss for each whether it is about syntactic, semantic or pragmatic aspects of the language.

Q8 (bonus points) In your own words, explain the concept of *definite assignment*. Why is it useful and why is it not perfect? Make a comparison with C and/or C++. How does definite assignment relate to the `final` keyword?

Classes and inheritance

In object-oriented languages, classes form a hierarchy with a parent class being extended by sub-classes that can themselves be extended. The following questions relate to classes and the concept of inheritance.

Q9 (10 points) If a language supports multiple inheritance, a class can inherit methods and variables from multiple parent classes. However, if multiple parents define the same method, this causes ambiguity in determining which of the versions of the method is actually inherited. Does Java support multiple inheritance? If so, how is the problem of ambiguity resolved or addressed? Explain your answer

Q10 (20 points) What is the difference between a method declared with the `static` keyword and one without the `static` keyword within the same class? Refer in your answer to *instance variables* and explicitly address both syntactic, semantic and pragmatic differences.

Q11 (10 points) In Java it is possible to define classes within classes (i.e. to *nest* class definitions). Such classes are referred to as *inner classes*. What kinds of inner classes exist? Explain why inner classes can be useful

Q12 (bonus points) In your own words, explain the *overloading* concept in Java.

Q13 (bonus points) In your own words, explain the *overriding* concept in Java. How does it relate to the `final` keyword?

Additional bonus questions

Q14 (bonus points) Write the shortest syntactically valid Java program you can come up with that sends something to the standard output.

Q15 (bonus points) Write the shortest syntactically valid Java program you can come up with.

Q16 (bonus points) Enum declarations simplify working with types of enumerable values. Enum declarations introduce normal classes (and are in that sense redundant) but add certain conveniences to the programmer. What functionality is generated by the Java compiler for Enum declarations? How do Enum declarations simplify working with enumerable types for the programmer?