Fundamentals of Java

Lecture 1. Introduction to Java

Course information

- Lecturer: Ryan Heise
  Email: ryan@ryanheise.com
  Email subject must be prefixed with [FOJ]

- Classes are every Wednesday, 6-9pm
  Room: 10.02.340
  Dates: 26/02/14 - 28/05/2014
  No class on 23/04/2014
  (13 weeks)

Delivery

- Lectures + Laboratory exercises
  (Often intermixed)
- Programming Assignment

  Two kinds of certificates are awarded:

- Certificate of Attendance awarded for 80%+ attendance.
- Certificate of Completion awarded for completion of the assignment (and 80%+ attendance).

Background

Programs are represented by code.

- Code intended to be read by a human is expressed in a high-level language. e.g. Java
- Code intended to be read by a computer is expressed in a low-level language. e.g. Binary machine code.

Programs are usually written by humans in a high-level language, and then translated (compiled) into binary machine code.

Background

Problem 1: Different CPUs use different machine code languages.

Problem 2: Different Operating Systems provide different sets of features to programs.

Typically you have to build a different version of your program to run on each system that you want to support.

Background

Java's philosophy:

Write once, run anywhere.

Java introduced the Java "virtual" machine (JVM) to solve this problem. The JVM:

- is a piece of software that simulates a hypothetical machine.
- provides a "universal" machine language.
- has been ported to every major "real" machine.

Background

Programs are:

- written in the high level Java language
- compiled to Java Virtual Machine language

This compiled code will now run on any JVM (which runs on any major platform).

i.e. Write a Java program once, and it will run anywhere.

Let's begin!

Today's concepts:

- Objects
- Classes
- Methods
- Fields
- Statements
- Expressions
- Operators
- Comments

A program is a sequence of instructions

```
instruction1;
instruction2;
instruction3;
instruction4;
instruction5;
instruction6;
instruction7;
```

Instructions in Java are called ***statements***.

Different kinds of statement

```
int a;
int b;
a = 3;
b = 4;
int result;
result = a + b;
System.out.println(result);
```

How many different kinds of statement do you see?

Different kinds of statement

```
int a;    <-- Variable declaration
int b;
a = 3;    <-- Variable assignment
b = 4;
int result;
result = a + b;
System.out.println(result);
                  <-- method invocation
```

How many different kinds of statement do you see? -- 3 kinds.

Variable declarations

A variable is a storage location inside RAM.

A variable declaration statement allocates space for storing a value:

```
int age;
```

A variable declaration has 2 parts:

1. A type (e.g. "int", short for integer)
2. A name (e.g. "age")

Variable types

The type restricts what type of value can be stored into that variable.

```
int age;
```

This variable can store a 32 bit integer.

```
long milliseconds;
```

This variable can store a 64 bit integer.

Variable types

What are the primitive types?

boolean	**binary value (either true or false)**
byte	8 bit integer
short	16 bit integer
int	**32 bit integer**
long	64 bit integer
float	32 bit real number
double	**64 bit real number**
char	**16 bit UNICODE/ASCII character**

Most common types in bold.

Variable assignments

A variable assignment statement stores a value into a variable, replacing what's already there.

```
age = 27;
```

- The left-hand side must be the name of a variable already declared.
- The right-hand side must be a value of the appropriate type.

Variable assignments

A variable assignment statement stores a value into a variable, replacing what's already there.

age = 20 + 7; ← this will also work

The value 27 will be computed first, and then the value 27 will be stored into variable age.

Values

Values are expressed using **expressions**.

Every value has a type.

Expression	Type	Value
10 + 5		
(10 + 5) * 2		
10		
9 / 2		
9 % 2		
9.0 / 2.0		
1 / 2.0		

Values

Values are expressed using **expressions**.

Every value has a type.

Expression	Type	Value
10 + 5	int	15
(10 + 5) * 2	int	30
10	int	10
9 / 2	int	4
9 % 2	int	1
9.0 / 2.0	double	4.0
1 / 2.0	double	0.5

Shortcuts and variations

Declare a variable AND assign to it at once:

```
int x = 3;
```

Declare multiple variables in one line:

```
int x, y, z;
```

Declare multiple variables and initialise some:

```
int x, y = 3, z;
```

Variables example

```
int x, y, z;  <-- declare 3 int variables
x = 3;
y = 5;
z = (x + y) / 2;
x = x + z;
System.out.println(x);
```

What does this program print?

Variables example

```
int x, y, z;  <-- declare 3 int variables
x = 3;
y = 5;
z = (x + y) / 2;
x = x + z;
System.out.println(x);
```

Answer: 7

Type checking

A variable can only accept values of the designated type.

```
int x;
boolean b;

x = 3;      <-- OK
x = true;   <-- ERROR
b = true;   <-- OK
```

Strings

The String type represents a string of characters (e.g. a word, a sentence, or random characters).

```
String name;

name = "Ryan Heise";

System.out.println(name);
```

String addition

The "+" operator for strings joins two strings together.

```
String s1 = "Mary";
String s2 = "Anne";
String result = s1 + s2;

System.out.println(result);
```

Will print: `MaryAnne`

Exercises

1. By using a separate variable for each intermediate calculation, write a program that calculates the volume of a house with wall dimensions 10x5x3m and with a roof extending an additional height of 2.5m in the shape of a triangular prism.
2. In a similar way, write a program that calculates the circumference and volume of a wheel with radius 30cm and width of 18cm.

Objects

A primitive is the smallest type of value: e.g. a single integer, a single character, etc...

An object is a larger value that is built out of smaller values.

e.g. a "Rectangle" object may be built from 4 primitive values: x, y, width, height.

These are called the "data fields" of the object.

Objects

But an object is more than just a collection of data fields.

An object also provides methods for interacting with it, and for manipulating its data.

Thus, an object can be defined as a collection of fields, and methods for interacting with and manipulating those fields.

Object-oriented programming

An object-oriented program is a collection of objects that interact with each other.

In a banking application, there may be millions of bank account objects, each containing data fields such as the bank balance, account holder, etc.

In a car racing game, there may be a number of car objects, each with data fields such as the x, y position and current speed and direction

Classes

The programmer must write the code on which the objects run. However,

- each bank account does not need to have different code.
- each car object does not need to have different code.

Similar types of objects are grouped into "classes". The programmer needs only to write code once for each "class".

Note: A class name should begin with an uppercase letter.

Classes are types, too

Suppose the programmer defines class "Car" containing the code for all car objects. "Car" is not only a class, it is also a type.

Therefore, the variable:

```
Car herbie;
```

allocates a variable of type "Car", i.e. a variable in which we are allowed to store a car object.

Creating an object

Objects are created from classes via the "new" operator:

```
herbie = new Car();
```

This line creates a new object from class Car and stores the resulting object into variable "herbie".

Interacting with an object

We interact with objects via the methods that it provides to us. To know what methods are supported, we need to look at its class definition.

```
herbie.drive(100);
```

This line invokes herbie's "drive" method with argument 100. The car "herbie" will drive forward 100 metres.

A simple program

```
Car herbie;

herbie = new Car();

System.out.println("Initial position: " + herbie.position);
herbie.drive(100);
herbie.drive(50);
System.out.println("Final position: " + herbie.position);
```

Exercises

Open a copy of the Shapes example project of BlueJ.

1. In the BlueJ code pad, create a new circle and try to interact with it using every one of its methods.
2. Write a program that creates various shapes and manipulates them to form a house.