UNIVERSITY OF PORTSMOUTH

OS Week 2 - Lab

Objects and Fields in Java

Introduction

Create your own OS cheat sheet. This sheet will include all the OS abbreviations/Terminologies shown on the OS lecture slides. Your tutor will explain how to do this. However, feel free to make it as you prefer. Your role is to fill this sheet while we are moving week by week.

Next week we will start running various Java programs to illustrate ideas about concurrency. As a prerequisite, we will review a little about the structure of Java - especially how objects and fields work in Java.

Many of you will be familiar with much of the following material. But go through the script here anyway, then do the exercise at the end and *make sure you are very clear about the difference between **static** variables and **instance** variables in Java*.

Assuming you are using Netbeans as introduced last week, here are some hints:

1. Don't be afraid to create new projects as you go - as a rule, you will want to create a new project for each example. Usually, there are several such examples each week, so you may create several projects per week. In the long run, this will save a lot of grief.
2. In the wizard for creating a new project, remember to change the name of the main class from `Main` to whatever class name is given in the lab scripts for the main class. You can usually accept the package name Netbeans gives you.
3. On the Netbeans *Run* menu, there is an option for setting the current main project. If necessary, use this to switch between successive projects.

Your lab book entry for each experiment should describe briefly what the experiment comprises of and what the outcome teaches us; it should also contain evidence that the experiment was actually run, typically in the form of a screenshot or just copy/pasted text of the output of the program from the Netbeans console. You won't get credit for simply reproducing the source code of the example codes in your lab book (although some people find it useful to include some source for their own records).

Remember to answer the reflective question, and include evidence of your attempts

at the exercises at the end of the lab script, where you are likely to need to give your source code.

Classes, Fields and Objects

Remember that a Java program contains one or more class definitions. A class contains named sections of code called ***methods***. One such section is the *main method*, which is the first method called to start a Java application.

Besides methods, a class can include ***fields***, which are just items of data. This class:

```
class MyClass {
      int a;
      String b;
}
```

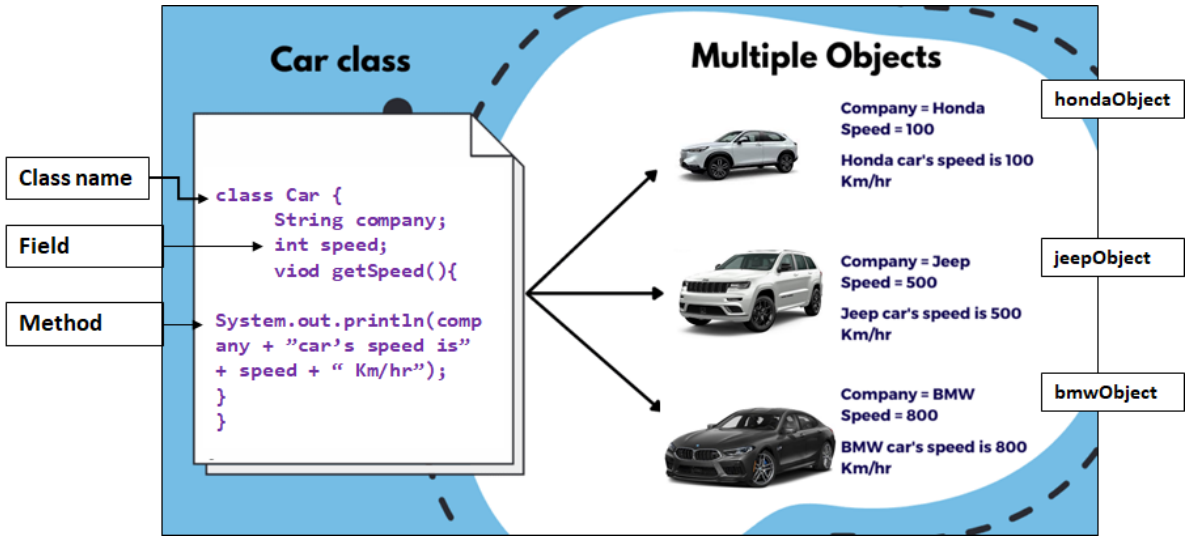
has no methods but two fields - a numeric (integer) variable called `a` and a string variable called `b`.

A class generally describes a *type* of object. To create an object of type `MyClass` we use the new operation:

```
MyClass myObject = new MyClass();
```

This line creates an object referred to here as `myObject`, which is a new object of type `MyClass`. The object contains its own copies of the variables `a` and `b`.

For example, you may have a class called `Class Car`. From this class, you can create multiple objects. Such as `hondaObject`, `jeepObject`, `bmwObject`,...etc. Each object has its own copies of the fields and methods.



Experiment 1

Use Netbeans to run a program like this:

```
public class MyObjectTest

{ public static void main(String args [])
   {
     MyClass object1 = new MyClass() ;
     // set a and b in object1
     object1.a = 42;
     object1.b = "my string value";
     MyClass object2 = new MyClass();

     // set a and b in object2
     object2.a = 23;
     object2.b = "my other string value";

     // print fields:
     System.out.println("Field a in object1 is: " + object1.a);
     System.out.println("Field b in object1 is: " + object1.b);

     System.out.println("Field a in object2 is: " + object2.a);
     System.out.println("Field b in object2 is: " + object2.b);
   }

}


class MyClass
{
     int a;
     String b;
}
```

Here we have two objects of the same type called `object1` and `object2`. Run this code and make sure you understand what it is doing.

Methods on objects

The `main()` methods defined in earlier examples were what are called *static methods* - they didn't refer to any particular object. In Java, it is more usual to work with *instance methods*, which are sections of code that operate *in the context of a particular object.*

Experiment 2

Here is a new, more "object-oriented" version of `MyObjectTest` in which we define an instance method called `printMyFields`. Notice that an instance method doesn't have the static keyword in its definition. This implies - by default - it is an instance method:

```
public class MyMethodTest
{
	public static void main(String args [])
	{
		MyNewClass object1 = new MyNewClass();
		object1.a = 42;
		object1.b = "my string value";

		MyNewClass object2 = new MyNewClass();
		object2.a = 23;
		object2.b = "my other string value";

		System.out.println("Fields of object1:");
		object1.printMyFields() ;

		System.out.println("Fields of object2:");
		object2.printMyFields() ;
	}
}

class MyNewClass
{
	int a;
	String b;

	void printMyFields()
	{
		System.out.println("Field a is: " + a);
		System.out.println("Field b is: " + b) ;
	}
}
```


Run this example.

Note that *within the definition* of `printMyFields` there is no need for an object prefix when the fields a and b are used. Compare with the print statements in the previous version. The object prefix is given when the method is called, instead.

Static variables

The last thing you will need to know about before we embark on concurrent programming next week is *static variables*.

Fields like `a` and `b` in `MyClass` are called *instance variables*. Every object of type `MyClass` has its own, distinct version of these variables, as we saw in the previous examples.

Java classes can also declare variables of which there is only a *single* copy for the whole class. These are called *static* variables, and they are identified by a keyword `static` in their declaration.

Experiment 3

Here is an example with a static field:

```
public class MyStaticVariableTest
{

	public static void main(String args [])
	{
		MyOtherClass.b = "my string value";

		MyOtherClass object1 = new MyOtherClass();
		object1.a = 42;

		MyOtherClass object2 = new MyOtherClass();
		object2.a = 23;

		System.out.println("Fields of object1:");
		object1.printMyFields();

		System.out.println("Fields of object2:");
		object2.printMyFields();
	}
}

class MyOtherClass
{
```



```
	int a;
	static String b;
	void printMyFields()
	{
		System.out.println("Field a is: " + a);
		System.out.println("Field b is: " + b);
	}
}
```

Again, run this example, and make sure you understand what is going on. Note well that here we only initialize the static field `b` once, and its prefix is a class name rather than an object name.

In the method, the same value is "seen" by all the objects.

The difference between ordinary fields ("instance variables") and static variables is subtle, but you should make sure you understand it. Later we will use static variables as a simple way of *sharing* variables between different parts of a program - especially between different threads.

Reflective Question

Answer questions like this in your lab book.

What would happen if an instance method in (say) `MyOtherClass` (Experiment 3) assigned a new value to the variable `a`? Would the change affect `a` in any other object? How about if an instance method called on one object changed the value of `b` - would that change affect the `b` value read by other objects?

Exercises

1. Let's recreate the scenario in the first part of the reflective question. Starting from the code in Experiment 3, add a new instance method to `MyOtherClass` that changes the value of `a` (assigns some new distinct value to it). Change the main method to call your new method on just `object1`, after all fields have been initially set, and before the calls to `printMyFields()` on both objects. Note the outcome.

2. Now the scenario in the second part of the reflective question. Add another new instance method to `MyOtherClass` that similarly changes the value of `b`

(hint: you don't need any prefixes), call it method on `object1`, then call `printMyFields()` on both objects.

Are your results consistent with your answer to the reflective question?

SPECIAL TOPIC: Running inline assembler code in GNU C

1. Go to AppsAnywhere and search for the Code::Blocks IDE. Start it running. Click through the screen concerning compiler detection (GCC should be detected).
2. Click on "File" -> "New" -> "Project", and select "Console Application".
3. Select the language as "C".
4. Choose a Project title (I used "asm-egs"). You can set "Folder to create project in" as "N:\".
5. Accept defaults on next screen, and click "Finish".
6. On the file chooser at the left of the screen, right-click on project folder (e.g. "asm-egs"), select "Build options..." from the drop down, then select "Other compiler options", and type "-masm=intel" in the text box, to select the assembler dialect.
7. On the file chooser, go to "Sources" and double click on "main.c".
8. Replace the ready-made "Hello World" program with this code:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{

      int result ;
      // Run assembly code from week 1 lecture, slide 16

      asm(
      "MOV EBX, EAX ;"
      "ADD EBX, 4 ;"

      : "=b" (result)
      : "a" (27)
      :
      );
      printf("Result of addition = %d\n", result);

      return 0;
}
```

9. Click the run button (green triangle) to optionally **build and run** the project.

The first string in an `asm` block contains the assembly language instructions to be executed. The clauses starting with colons define various inputs and outputs and side effects (e.g. the shorthand "a" (27) means the register EAX will be initialized to 27 before starting this block of instructions).

The program above should print the final value of the EBX register.

Assuming the code above works as expected, try replacing the asm block part only with this and leave the rest:

```
// Run assembly code from week 1 lecture, slide 18
asm(
"MOV ESI, 105672 ;"
"MOV EAX, [ESI] ;"

: "=a" (result)
:
: "esi"
) ;
printf("Result read from memory = %d\n", result);
```

This code may or may not run successfully. Here we are reading from an essentially random address 105672. If, on your machine, this address is readable, the program should succeed and print out a pretty random number - the contents of memory at this address. But this address could be in a page of memory that hasn't been allocated or that is read protected, in which case the program may fail with some kind of "access violation" (code 0xc0000005 in Windows).

Now replace the `asm` block with this:

```
// Run assembly code adapted from week 1 lecture, slide 20

asm(
"MOV DX, 368 ;"
"IN EAX, DX ;"

: "=a" (result)
:
: "dx"
) ;
printf("Result read from port = %d\n", result);
```


This time when you run the code (on Windows) it should fail with an error code like this:

```
0xC0000096
```

Look this code up here:

http://errorco.de/

What is going on here? There is a list of Windows recognized hardware exception codes here:

https://docs.microsoft.com/en-us/cpp/cpp/hardware-exceptions

