INFO1103: Introduction to Programming

School of Information Technologies, University of Sydney

Week 7: Classes and Objects

We will cover: What is a class, instance variables, encapsulation, `public` and `private`, creating and using Objects, reference type, methods, the `this` keyword

You should read: §§5.1 – 5.3, 6.1

Lecture 13: Classes and Objects

Defining classes. Creating and using Objects

Types and Values

A type is a kind of a thing e.g. laptop is a type of computer, 3.14 is a type of a real number, tulip is a type of a flower.

Both `int` and `double` are *types* in Java.

data type	**possible values**	**operations**
`boolean`	`true`, `false`	not, and, or, Xor
`int`	$-2^{31} \ldots 2^{31} - 1$ integer numbers ($\mathbb{Z}$)	+, - ×, /
`double`	2^{64} possible real numbers ($\mathbb{R}$)	+, - ×, /

With these types you can do many operations from anywhere in the program

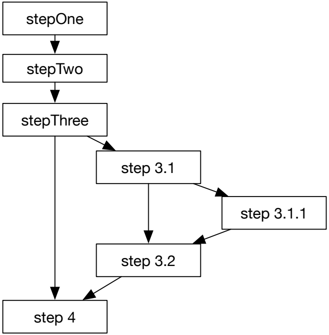
Reading or modifying a value of a type

```java
int answer = 1; // must always be positive
int[] input = getInput();

answer = 100 * input[0]; // step 1

if ( input[1] % 2 == 0 ) // step 2
   input[2] += answer;

while (input[3] > 0) // step 3
{
   if (input[3] <= 0)
      break;
   if ( input[2] > 5 ) // step 3.1
   {
      input[2] -= 1;
      if (answer <= 0)  // step 3.1.1
         answer += 1;
   }
   answer = answer + 1; // step 3.2
   input[3] -= 1;
}
answer = calcMagicFlux( answer, 10000, 48.9 ); // step 4
System.out.println("Answer: " + answer);
```

Reading or modifying a value of a type

Organised into methods

```java
int answer = 0;
int[] input = getInput();
answer = stepOne(input[0]); // do all steps
// calls -> answer = stepTwo(answer, input[1]);
//   calls -> answer = stepThree(answer, input[2], input[3]);
//     calls -> answer = stepFour( answer, 100 );
System.out.println("Answer: " + answer);
```



- When is `answer` required for read access?
- When is `answer` required for write access?
- ...
- If each task is done one of 4 persons, how do you control when answer is read or write?

Objects

We use the primitive types in programming all the time, but there are cases when we need types which:

- control access to the value
- control the operations possible with the value
- contain more than one kind of value (not an array, but a composite of other types)

An object is a thing that may have data or particular methods, or (most commonly) *both.* them a bit differently.

Objects have Class

The *type* of an object variable is its *class*.

```
Point p;
```

Objects are *instances* of things of a particular *Class*.

```
Point topleft = new Point(-1 , -1);
Point right = new Point(1, 0);
Point home = new Point( -3388797, 15119390 );
```

Simple Classes

Let’s make a very simple class with a primitive type

```java
public class PositiveInteger {
   public int number;
}
```

We make an *instance* of this class like this:

```java
      PositiveInteger num = new PositiveInteger();
```

This calls a special “method” called the *constructor*.

Constructors

The constructor does the work of building an instance of the class.

It creates space in memory for the object when it is called.

The constructor is called by the `new` operator

If there is no constructor defined, Java will automatically create a default constructor for the class

```java
public class PositiveInteger {
   public int number;
   public PositiveInteger() {}
}
```

Constructing a PositiveInteger object

We could have a very simple constructor for our new PositiveInteger class, like this:

```java
    public PositiveInteger() { }
        // this constructor does nothing!
```

or this:

```java
    public PositiveInteger() {
        number = 1;
    }
```

or this:

```java
    public PositiveInteger(int initialValue) {
        number = initialValue;
    }
```

Constructors

A *constructor* for a *class* is a plan, or blueprint, to make an instance of the class.

When you make classes yourself you often need to make instances of them

What happens when you call the constructor?

Creating multiple instances.

```
1       PositiveInteger eyes = new PositiveInteger();
2       PositiveInteger kelvin = new PositiveInteger();
3       //
```

- What is happening at each step?
- How much memory is used?
- What would a diagram look like to describe the memory at line 3?

Creating an instance

When you do that,

- space is made for the object of type `PositiveInteger`
- the instance variable of the class is set to some value
- variables `eyes` and `kelvin` are ready for use.

What do you notice?

- no return type — it returns an object;
- can have more than one constructor — just like overloading a method;
- you use the `new` keyword to call the constructor:

```
    PositiveInteger n = new PositiveInteger();
    PositiveInteger y = new PositiveInteger(47);
```

Using the instance

With an Object variable we use the dot . to refer to `public` methods e.g.

```java
String text = "beetle";
int length = text.length();
```

We can do the same with `public` instance variables, with the parentheses

```java
PositiveInteger chairs = new PositiveInteger();
chairs.number = 50;
int value = chairs.number;

System.out.println( "value: " + value );
System.out.println( "chairs.number: " + chairs.number );

chairs.number = 16; // change which part of memory?

System.out.println( "value: " + value );
System.out.println( "chairs.number: " + chairs.number );
```

The `public` keyword is an access modifier. The value is visible from anywhere in the program.

Building larger programs

One program can have many classes[1]

Here is a file `PositiveInteger.java`

```java
public class PositiveInteger {
   public int number;  // visible everywhere
}
```

Here is a file `MainProgram.java`

```java
public class MainProgram {
   public static void main(String[] args) {
      PositiveInteger persons = new PositiveInteger();
      persons.number = 50;
      System.out.println( "persons.number: " + persons.number );
   }
}
```

```
~> javac PositiveInteger.java MainProgram.java
~> java MainProgram
```

[1] There is only one main method

Access modifier: public

public instance variables are not useful if we want to restrict read/write access

By design, any object of type PositiveInteger should contain positive integer values

public allows any part of the program to modify the value without checking it is correct

Modified MainProgram.java

```
public class MainProgram {
    public static void main(String[] args) {
        PositiveInteger persons = new PositiveInteger();
        persons.number = -470;
        System.out.println( "persons.number: " + persons.number );
    }
}
```

Access modifier: private

The private keyword is an access modifier. The value is only visible within the scope of the class.

```java
public class PositiveInteger {
   private int number; // only visible in scope of this class
}
```

```java
public class MainProgram {
   public static void main(String[] args) {
      PositiveInteger persons = new PositiveInteger();
      persons.number = -470;
      System.out.println( "persons.number: " + persons.number );
   }
}
```

```
~> javac PositiveInteger.java MainProgram.java
```

Object Methods

We can create many instances of the same class, but they each have their own memory

We can define a method of a class. A method associates with the memory of the object

```java
String animal = "Emu";
String place = "Rainbow Valley";
int aLen = animal.length();
int pLen = place.length();
```

The two String objects each have the same method, but the method operates on different data, unique to the String object.

Read Access: Get Methods

Methods associate with the memory of an object

A private instance variable is only visible within the scope of the class

Thus, any method defined within the class can both read and write private instance variables

```java
public class PositiveInteger {
    private int number; // only visible in scope of this class

    private void dostuff()
    {
        number = 34; // can change, within scope
    }
}
```

Read Access: Get Methods

For an outside class to have access to a private value, it needs to call a public method

If we want to make that value read only, the method returns a copy of the value

```java
public class PositiveInteger {
   private int number; // only visible in scope of this class

   public int getNumber() // visible everywhere
   {
      return number; // return a copy of the value
   }
}
```

No outside class can change this value!

Changing private instance variables

We could use a constructor to initialise that value when the object is created.

```java
public class PositiveInteger {
    private int number; // only visible in scope of this class

    public PositiveInteger(int initialValue) {
        number = initialValue;
    }

    public int getNumber() // visible everywhere
    {
        return number; // return a copy of the value
    }
}
```

When does the value change over the lifetime of the object?

Write Access: Set Methods

Another public method. This takes parameters and changes the value

```java
public class PositiveInteger {
    private int number; // only visible in scope of this class

    public PositiveInteger(int initialValue) {
        number = initialValue;
    }

    public void setNumber(int newValue) // visible everywhere
    {
        number = newValue; // change to another value
    }
    public int getNumber()
    {
        return number;
    }
}
```

Controlling possible operations

Instead of public access to the instance variable, we use methods

Modified `MainProgram.java`

```java
public class MainProgram {
   public static void main(String[] args) {
      PositiveInteger persons = new PositiveInteger();
      persons.setNumber( -470 ) ;
      System.out.println( "persons.number: " + persons.getNumber() )
   }
}
```

Isn’t this the same problem? can we prevent -470?

Controlling possible operations

The set method we define is the only way the value can change from anywhere in the program

We can define restrictions on what possible values it can have

```
// allowable number: zero or greater
public void setNumber(int newValue)
{
   if ( number >= 0)
      number = newValue;
}
```

Controlling possible operations

We can further restrict how the values can change by having specific operations instead of `setNumber()`

```java
public class PositiveInteger {
   private int number; // only visible in scope of this class

   public PositiveInteger(int initialValue) {
      number = initialValue;
      if (initialValue < 0) // always keep positive
         number = 0;
   }
   // increment the number by one
   public void incrementNumber() {
      number = number + 1;
   }
   // decrement the number by one
   public void decrementNumber()
      if (number < 0) // always keep positive
         number = number - 1;
   }
   public int getNumber() {
      return number;
   }
}
```

Controlling possible operations

Modified `MainProgram.java`

```java
public class MainProgram {
    public static void main(String[] args) {
        PositiveInteger persons = new PositiveInteger(2);
        persons.decrementNumber();
        persons.incrementNumber();
        persons.decrementNumber();
        persons.decrementNumber();
        persons.decrementNumber();
        persons.incrementNumber();
        persons.decrementNumber();
        System.out.println( "persons.number: " + persons.getNumber() )
    }
}
```

Desk check the instance variable of `persons`