5 FIELDS AND METHODS

Terry Marris 10 April 2001

5.1 OBJECTIVES

By the end of this lesson the student should be able to understand

- the difference between attributes and fields
- the difference between operations and methods
- the difference between parameters and arguments
- the difference between a message and a return value

5.2 PRE-REQUISITES

The student should be comfortable with simple Java classes, class diagrams, attributes and operations. The student should have used *int* and *double* variables in simple programs and understand the term information hiding.

5.3 PREVIEW

In this lesson we design a class by drawing a class diagram, supported by descriptive prose, showing WHAT a class does; we implement a class by specifying HOW the operations are actually carried out; we test the implementation by using its methods and looking for errors in them. We introduce the concepts message and receiver, argument and parameter.

We specify a simple money class, implement it in Java and then show how it may be used.

5.4 SPECIFICATION

What the Money class does is described below.

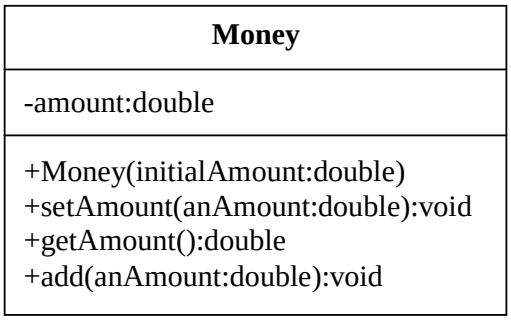


Figure 5.1 The Money Class

The class is named *Money*. It has a private attribute named *amount*. *amount* stores a currency value. For example, *amount* could hold 50.0 to represent £50.00.

The class has a public constructor *Money(initialAmount:double)*. The constructor initialises the private attribute, *amount*, with an *initialAmount*.

The *Money* class has three public operations:

+*setAmount(anAmount:double):void*	stores *anAmount* in *amount*
+*getAmount():double*	retrieves the value stored in *amount*
+*add(anAmount:double):void*	increases the value stored in *amount* by *anAmount*

5.5 IMPLEMENTATION

5.5.1 FIELDS

The class is named *Money*. It has a private attribute named *amount*.

```
public class Money {
  private double amount;
```

Attributes are known as fields in Java.

5.5.2 CONSTRUCTORS

The class has a public constructor *Money(initialAmount:double)*. The constructor initialises the private attribute, *amount*, with an *initialAmount*.

```
public Money(double initialAmount)
{
  amount = initialAmount;
}
```

Whatever value is stored in *initialAmount*, that is assigned to the *amount* field. How does *initialAmount* get its value? We shall see how when we come to the Usage section that follows.

5.5.3 SETTING METHODS

The public operation *setAmount(anAmount:double):void* stores *anAmount* in *amount*.

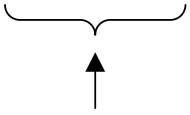
```
public void setAmount(double anAmount)
{
  amount = anAmount;
}
```

Whatever value is stored in *anAmount*, that value is assigned to the *amount* field.

Operations are known as methods in Java.

double anAmount is an example of a parameter. A parameter is like a local variable - it can be used only by the method in which it is defined.

```
public void setAmount(double anAmount)
```



parameter

A setting method always has a single parameter. It assigns a value to a single field, and does nothing else. A setting method is just for one field.

5.5.4 GETTING METHODS

The public operation *getAmount():double* retrieves the value stored in *amount*.

```
public double getAmount()
{
  return amount;
}
```

return is a Java keyword. The value stored in the *amount* field is retrieved and made available to the client - see the Usage section that follows.

Getting methods never have parameters. They always return the value stored in a single field and never do anything else. A getting method is for just one field.

5.5.5 ADD METHOD

The public operation *add(anAmount:double):void* increases the value stored in *amount* by *anAmount*.

```
public void add(double anAmount)
{
  amount = amount + anAmount;
}
```

The value stored in the *amount* field is replaced by a new value. This new value is the original contents of *amount* added to the parameter *anAmount*.

5.6 USAGE

We use the *Money* class in a *main()* method.

```
public static void main(String[] s)
{
```

5.6.1 USING THE CONSTRUCTOR

We create a new *Money* object named *fund*. Initially, the value assigned to the *amount* field is 0.0.

```
Money fund = new Money(0.0);
```

0.0 is an example of an argument value. It is this value that is passed to the constructor's parameter *initialAmount*.

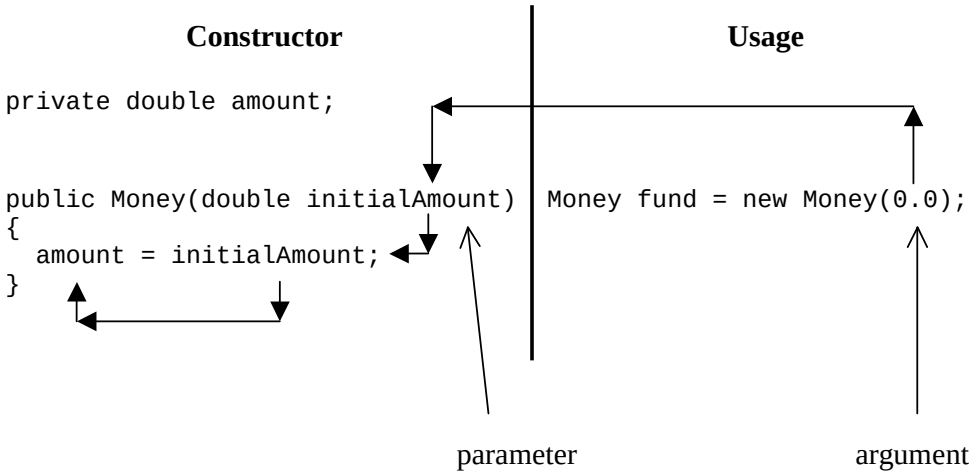


Figure 5.2 *Argument values are passed to parameters*

new is a Java operator. It returns the address in memory of where a newly created object is located. This address is assigned to *fund*. So, *fund* is a reference to a money object stored in memory. But we conveniently think of it as being the object.

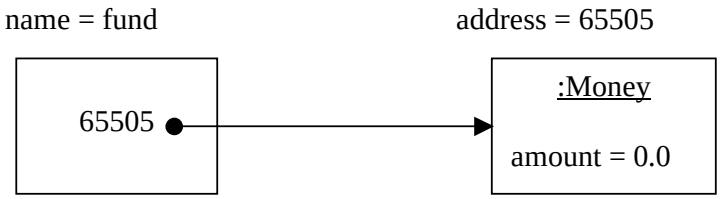


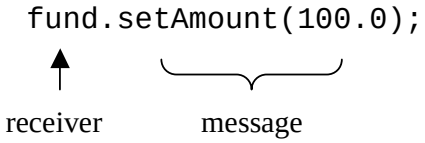
Figure 5.3 *fund is a reference to a Money object in memory*

5.6.2 USING THE SETTING METHOD

To put 100.0 into the *amount* field, we use the setting method for the *fund* object like this:

```
fund.setAmount(100.0);
```

setAmount(100.0) is known as the message. We are sending the message *setAmount(100.0)* to the *fund* object. The *fund* object then is known as the receiver.



The *fund* object looks through all its method signatures (Method signatures are just the method headings.)

```
public Money(double)
public void setAmount(double)
public double getAmount()
public void add(double)
```

If it finds a method signature that matches the message *setAmount(100.0)* that method is executed. If it does not find a method signature that matches the message, a *methodName(parameterType) not found* error is generated.

5.6.3 USING THE GETTING METHOD

The getting method *double getAmount()* retrieves and returns the value stored in the *amount* field. We send the message *getAmount()* to the *fund* object.

```
fund.getAmount();
```

The *getAmount()* method returns the contents of the *amount* field. What can we do with a value that is returned?

1 We could assign it to a variable.

```
double contents = fund.getAmount();
```

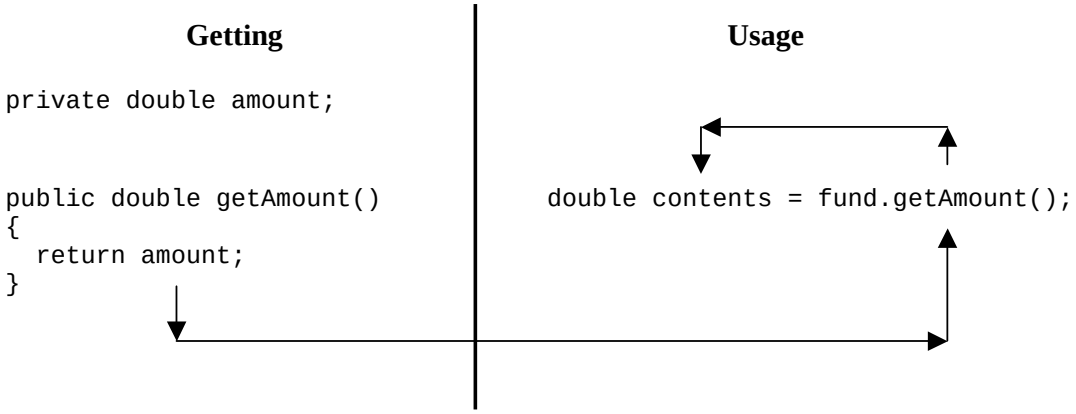


Figure 5.4 *how the value stored in the private field amount is retrieved*

2 We could display it on the screen.

```
System.out.println(fund.getAmount());
```

3 We could ignore it and do nothing with it. But this would not be of any use.

```
fund.getAmount();
```

In the method signature *double getAmount()*, *double* is known as the method return type.

The value returned by the message *fund.getAmount()* is the value that is retrieved from the *amount* field by the *getAmount()* method.

5.7 TESTING THE MONEY CLASS

Shown below is the *Money* class implementation, a *main()* method to test the class, and the result of running the *main()* method.

```
/* Money.java
   Terry Marris  10 April 2001
*/

public class Money {
  private double amount;

  public Money(double anAmount)
  {
    amount = anAmount;
  }

  public void setAmount(double anAmount)
  {
    amount = anAmount;
  }

  public double getAmount()
  {
    return amount;
  }

  public void add(double anAmount)
  {
    amount = amount + anAmount;
  }

  public static void main(String[] s)
  {
    Money fund = new Money(0.0);
    System.out.println("To begin with, the fund contains £" +
                        fund.getAmount());

    fund.setAmount(100.0);
    System.out.println("After putting £100.00 in, " +
                       "the fund now contains £" +
                        fund.getAmount());

    fund.add(25.75);
    System.out.println("After adding a further £25.75, " +
                       "the fund now contains £" +
                        fund.getAmount());
  }
}
```

Output from the program run:

```
To begin with, the fund contains £0.0
After putting £100.00 in, the fund now contains £100.0
After adding a further £25.75, the fund now contains £125.75
```

EXERCISE Identify and label the statement in the main() method that produced each line of the output shown above.

Some points to notice:

1 Including a *main()* method in a class provides a convenient way to test each method.

2 Notice how we placed line breaks when our *System.out.println()* statements looked like they were about to go off the edge of the screen or page (at about column 70). For example

```
System.out.println("After putting £100.00 in, " +
                   "the fund now contains £" +
                    fund.getAmount());
```

We terminated the string at a convenient place, then added on to it the next bit.

3 Notice the neat and tidy layout.

4 In testing the class we used each of its methods and checked that they all gave consistent and correct results.

Why did we not write

```
System.out.println(fund.setAmount(100.0));
```

Look at the *add(double)* method on page five. Its return type is *void*. void is the empty type. It has no values. So nothing can be returned to be printed. In fact, this statement would generate a compile time error.

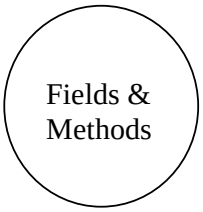
5.8 FURTHER READING

HORSTMANN C.S & CORNELL G *Core Java 2 Volume* 1 pp 132, 109

LEWIS J & LOFTUS W *Java Software Solutions* pp 52

ARNOW D & WEISS G *Java - An Object Oriented Approach* pp 8

5.9 REVIEW



fields

methods

- constructors
- setting
- getting
- parameters
- return type
- signature

clients

- new
- messages
- receivers
- arguments

5.10 EXERCISES

1 With the aid of examples explain the meaning of each of the terms shown below:

(a) fields **(b)** methods **(c)** setting methods

(d) getting methods **(e)** parameters **(f)** arguments

(g) messages **(h)** receivers **(i)** clients

(j) method signatures

2 Identify objects, receivers, messages and arguments in the code fragment shown below

```
Money  aSalary = new Money(25000.0);
aSalary.addAmount(3000.0);
```

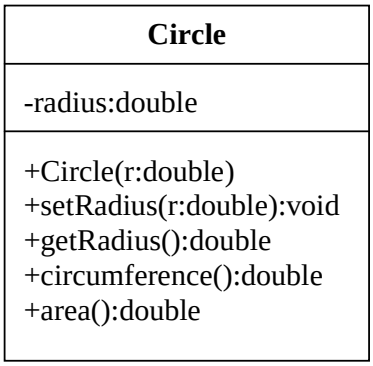
3 What is the usual purpose of a constructor? (It is not to create a new object!).

4 Write and test the method

+multiplyBy(anAmount:double):void

that multiplies the value stored in the *amount* field of the *Money* class by the given *anAmount*.

5 Implement and test the *Circle* class specified below.



Fields

private double radius	holds the radius of the circle.

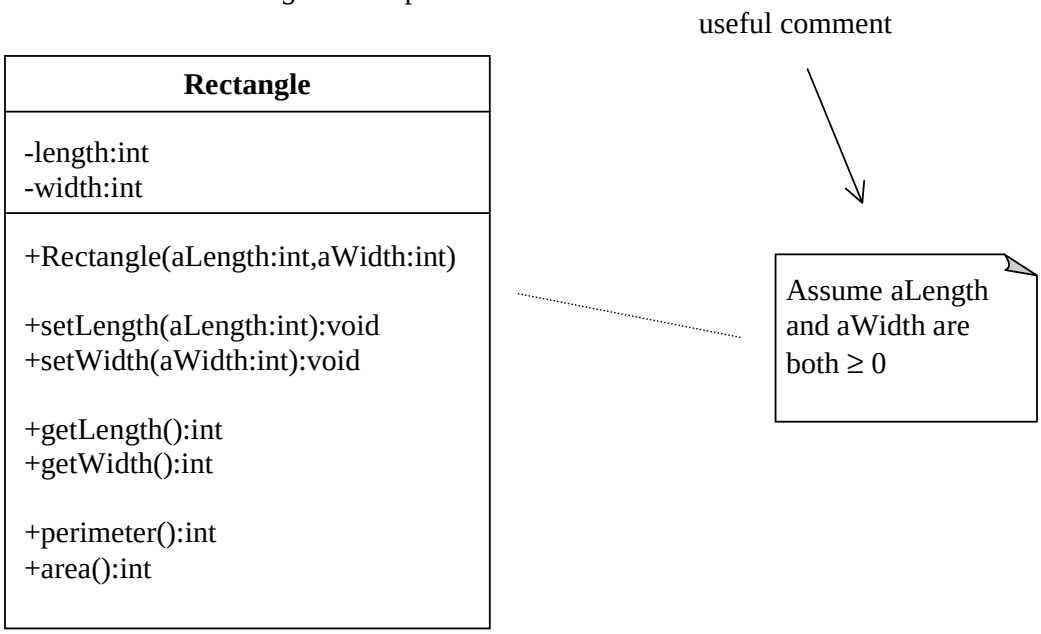
Constructors

public Circle(double r)	initialises radius with the given value, r.

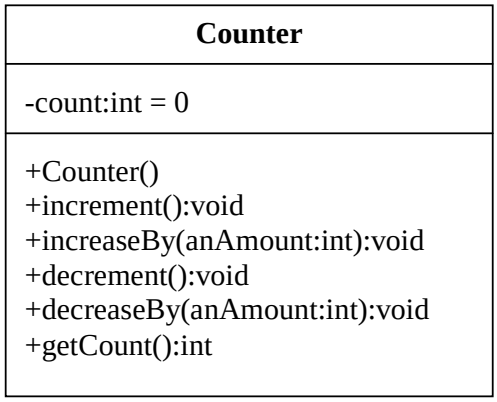
Methods

public void	setRadius(double r)	assigns the given value r to the radius.
public double	getRadius()	returns the value stored in radius
public double	circumference()	returns 2 * 3.1416 * radius
public double	area()	returns 3.1416 * radius^2

6 Implement and test the *Rectangle* class specified below.



7 Implement and test the Counter class specified below.



Fields

private int count	holds the current count value, default value zero.

Constructors

public Counter()	initialises count with the value 0.

Methods

public void	increment()	adds 1 to the value stored in count.
public void	increaseBy(int anAmount)	adds anAmount to the value stored in count.
public void	decrement()	subtracts one from the value stored in count.
public void	decreaseBy(int anAmount)	subtracts anAmount from the value stored in count.
public int	getCount()	returns the value stored in count.