5 Overloading

Introduction

This chapter will introduce the reader to the concept of method overloading

Objectives

By the end of this chapter you will be able to…

- Understand the concept of 'overloading'
- Appreciate the flexibility offered by overloading methods
- Identify overloaded methods in the online API documentation

This chapter consists of the following three sections:-

1) Overloading
2) Overloading To Aid Flexibility
3) Summary

Brain power
By 2020, wind could provide one-tenth of our planet's electricity needs. Already today, SKF's innovative know-how is crucial to running a large proportion of the world's wind turbines.
Up to 25 % of the generating costs relate to maintenance. These can be reduced dramatically thanks to our systems for on-line condition monitoring and automatic lubrication. We help make it more economical to create cleaner, cheaper energy out of thin air.
By sharing our experience, expertise, and creativity, industries can boost performance beyond expectations.
Therefore we need the best employees who can meet this challenge!
The Power of Knowledge Engineering
Plug into The Power of Knowledge Engineering.
Visit us at www.skf.com/knowledge
SKF

5.1 Overloading

Historically in computer programs method names were required to be unique. Thus the compiler could identify which method was being invoked just by looking at its name.

However several methods were often required to perform very similar functionality for example a method could add two integer numbers together and another method may be required to add two floating point numbers. If you have to give these two methods unique names which one would you call 'add()'?

In order to give each method a unique name the names would need to be longer and more specific. We could therefore call one method addInt() and the other addFloat() but this could lead to a proliferation of names each one describing different methods that are essentially performing the same operation ie. adding two numbers.

To overcome this problem in Java you are not required to give each method a unique name – thus both of the methods above could be called add(). However if method names are not unique the Java Runtime Environment (JRE) must have some other way of deciding which method to invoke at run time. ie. when a call is made to add(number1, number2) the machine must decide which of the two methods to use. It does this by looking at the parameter list.

While the two methods may have the same name they can still be distinguished by looking at the parameter list.:-

```
add(int number1, int number2)
add(float number1, float number2)
```

This is resolved at run time by the JRE. i.e. at run time the JRE looks at the method call and the actual parameters being passed. If two integers are being passed then the first method is invoked. However if two floating point numbers are passed then the second method is used.

Overloading refers to the fact that several methods may share the same name. As method names are no longer uniquely identify the method then the name is 'overloaded'.

5.2 Overloading To Aid Flexibility

Having several methods that essentially perform the same operation, but which take different parameter lists, can lead to enhanced flexibility and robustness in a system.

Imagine a University student management system. A method would probably be required to enrol, or register, a new student. Such a method could have the following signature…

```
enrollStudent(String pName, String pAddress, String pCoursecode)
```

However if a student had just arrived in the city and had not yet sorted out where they were living would the University want to refuse to enrol the student? They could do so but would it not be better to allow such a student to enrol (and set the address to 'unkown')?

To allow this the method enrollStudent() could be overloaded and an alternative method provided as…

```
enrollStudent(String pName, String pCoursecode)
```

At run time the JRE could determine which method to invoke depending upon the parameter list provided. Thus given a call to

```
enrollStudent("Fred", "123 Abbey Gardens", "G700")
```

the JRE would use the first method.

Activity 1

Imagine a method withdrawCash() that could be used as part of a banking system. This method could take two parameters:- the account identity (a String) and the amount of cash required by the user (int). Thus the full method signature would be:-

```
withdrawCash(String pAccountID, int pAmount).
```

Identify another variation of the withdrawCash() method that takes a different parameter list that may be a useful variation of the method above.

Feedback 1

An alternative method also used to withdraw cash could be withdrawCash(String pAccountID) where no specified amount is provided but by default £100 is withdrawn.

These methods essentially perform the same operation but by overloading this method we have made the system more flexible – users now have a choice they can specify the amount of cash to be withdrawn or they can accept the default sum specified.

Overloading methods don't just provide more flexibility for the user they also provide more flexibility for programmers who may have the job of extending the system in the future and thus overloading methods can make the system more future proof and robust to changing requirements.

Constructors can be overloaded as well as ordinary methods.

Activity 2

Go online and look at the Java Standard Edition API documentation by 1) going online to **java.sun.com/javase** 2) following the link to the API 3) selecting the link for Core API documents for the latest version.

At the time of writing this should take you to http://java.sun.com/javase/6/docs/api/

In the lower left panel you should see a long list of all the classes available to Java programmers. Scroll down this list until you find the String class. This will be quicker if you first select the Java.lang package in the upper left window (as the String class is in this package).

Look at the String class documentation in the main pane and find out how many constructors exist for this class.

Feedback 2

The String class specifies 15 different constructors. They all have the same method name 'String' of course but they can all be differentiated by the different parameters these methods require.

One of these constructors takes no parameters and creates an empty String object. Another requires a String as a parameter and creates a new String object that is a copy of the original.

By massively overloading the Sting constructor the creators of this class have provided flexibility for other programmers who may wish to use these different options in the future.

We can make our programs more adaptable by overloading constructors and other methods. Even if we don't initially use all of the different constructors, or methods, by providing them we are making our programs more flexible and adaptable to meet changing requirements.

Activity 3

Still looking at the String class in the API documentation find other methods that are overloaded.

Feedback 3

There are many methods that are not overloaded but there are also many that are. These include:- format(), indexOf(), replace(), split(), subString() and valueOf().

Looking at the different subString methods we see that we can find a substring by either specifying the starting point alone or by specifying start and end points.

When we use the subString() method the JRE will select the correct implementation of this method, at run time, depending upon whether or not we have provided one or two parameters.

5.3 Summary

Method overloading is the name given to the concept that several methods may exist that essentially perform the same operation and thus have the same name. The JRE distinguishes these by looking at the parameter list. If two or more methods have the same name then their parameter list must be different.

At run time each method call, which may be ambiguous, is resolved by the JRE by looking at the parameters passed and matching the data types with the method signatures defined in the class.

By overloading constructors and ordinary methods we are providing extra flexibility to the programmers who may use our classes. Even if these are not all used initially, providing these can help make the program more flexible to meet changing user requirements.