

FIT ICT Software Development

Lecture 5

Lecturer : Charles Tyner

RECAP.....

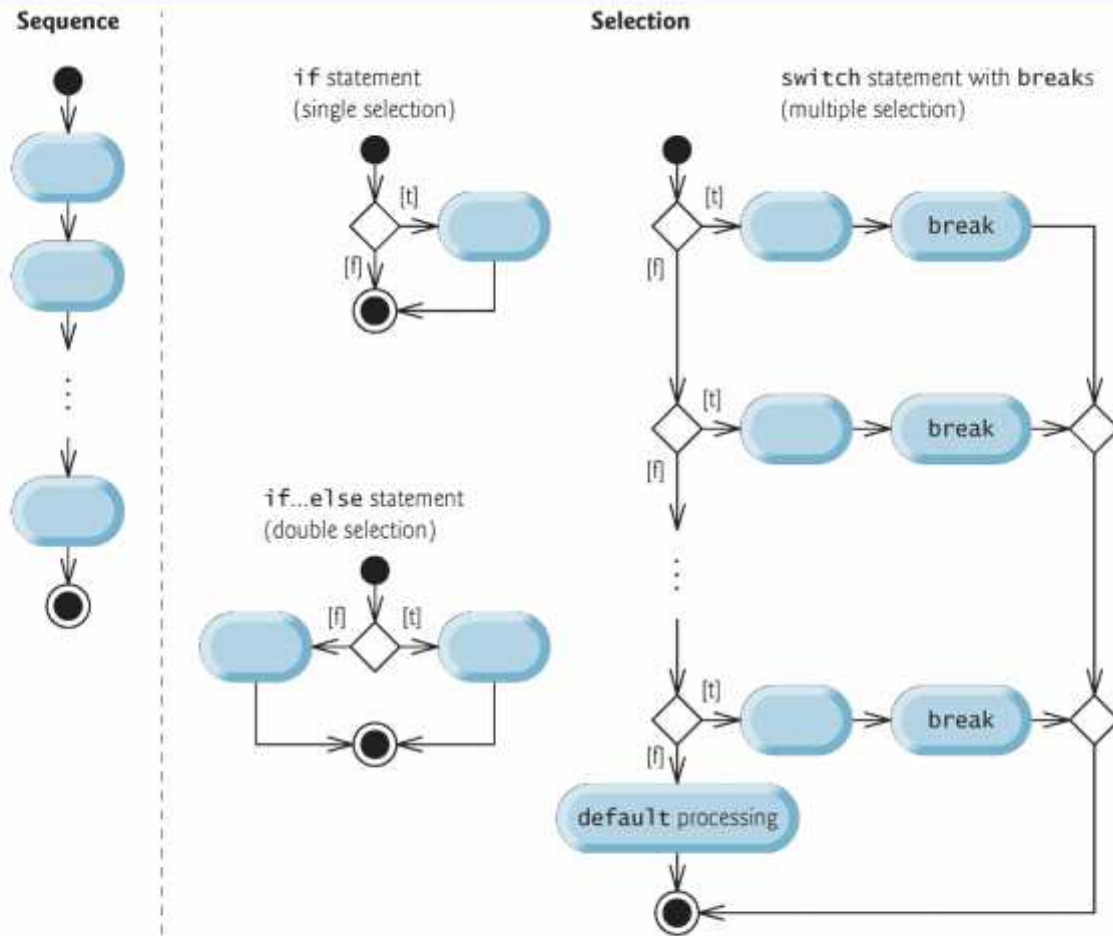


Fig. 4.19 | Java's single-entry/single-exit sequence, selection and repetition statements (Part 1 of 2)

Structured Programming Summary

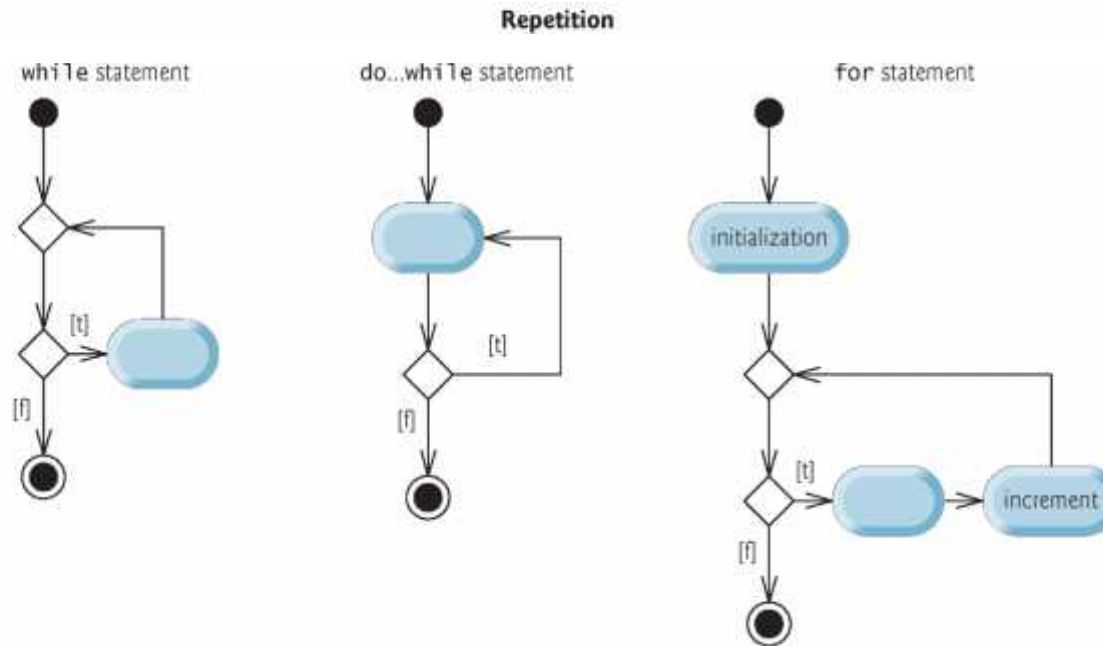


Fig. 4.19 | Java's single-entry/single-exit sequence, selection and repetition statements. (Part 2 of 2.)

Structured Programming Summary

- Structured programming promotes simplicity.
- Bohm and Jacopini proposed that only three forms of control are needed to implement an algorithm:
 - Sequence
 - Selection
 - Repetition
- The sequence structure is trivial. Simply list the statements to execute in the order in which they should execute.

Structured Programming Summary

- Selection is implemented in one of three ways:
 - `if` statement (single selection)
 - `if...else` statement (double selection)
 - `switch` statement (multiple selection)
- The simple `if` statement is sufficient to provide any form of selection—everything that can be done with the `if...else` statement and the `switch` statement can be implemented by combining `if` statements.

Structured Programming Summary

- Repetition is implemented in one of three ways:
 - `while` statement
 - `do...while` statement
 - `for` statement
- The `while` statement is sufficient to provide any form of repetition. Everything that can be done with `do...while` and `for` can be done with the `while` statement.

Introduction to Methods

Most computer programs that solve real-world problems are large.

Experience has shown that the best way to develop and maintain a large program is to construct it from small, simple pieces or modules.

“Take a large problem,
break it into smaller problems,
solve the smaller problems and
thereby solve the big problem”

Many key features of the Java language facilitate this approach to the design, implementation, operation and maintenance of large programs.

Modules in Java are called methods and classes.

Introduction to Methods

- Best way to develop and maintain a large program is to construct it from small, simple pieces.
 - called **divide and conquer**
- Methods facilitate the design, implementation, operation and maintenance of large programs.
- Normally, methods are called on specific objects
- **static** methods can be called without the need for an object of the class to exist

Program Modules in Java

- Java programs are written by combining new methods and classes that you write with predefined methods and classes available in the [Java Application Programming Interface](#) and in various other class libraries
- Related classes are typically grouped into packages so that they can be imported into programs and reused

Program Modules in Java

The Java API provides a rich collection of classes and methods for performing:

- common mathematical calculations, eg `math.pow`
- string manipulation,
- character manipulation,
- input/output,
- error checking,

... and many other useful operations.

This makes the programmers job easier because these methods provide many of the capabilities programmers need.



Software Engineering Observation 5.1

Familiarize yourself with the rich collection of classes and methods provided by the Java API (java.sun.com/javase/6/docs/api/). In Section 5.8, we present an overview of several common packages. In Appendix E, we explain how to navigate the Java API documentation. Don't reinvent the wheel. When possible, reuse Java API classes and methods. This reduces program development time and avoids introducing programming errors.

What are methods?

The programmer can write **methods** to define specific tasks that may be used at many points in a program.



These are referred to as programmer defined **methods**.

Program Modules in Java (cont.)

- The statements in method bodies are written only once, are hidden from other methods and can be reused from several locations in a program.

Using Methods promotes software reusability

- using existing methods as building blocks to create new programs
- Often, you can create programs from standardized methods rather than by building customized code
- Dividing a program into meaningful methods makes the program easier to **debug** and **maintain**.

Program Modules in Java (cont.)

- A method is **invoked** by a method call
- When the called method completes its task, it either returns a result or simply returns control to the caller
- Similar to the hierarchical form of management (Fig. 5.1)
 - A boss (the caller) asks a worker (the called method) to perform a task and report back (return) the results after completing the task
 - The boss method does not know how the worker method performs its designated tasks
 - The worker may also call other worker methods, unbeknown to the boss
 - This “hiding” of implementation details promotes good software engineering

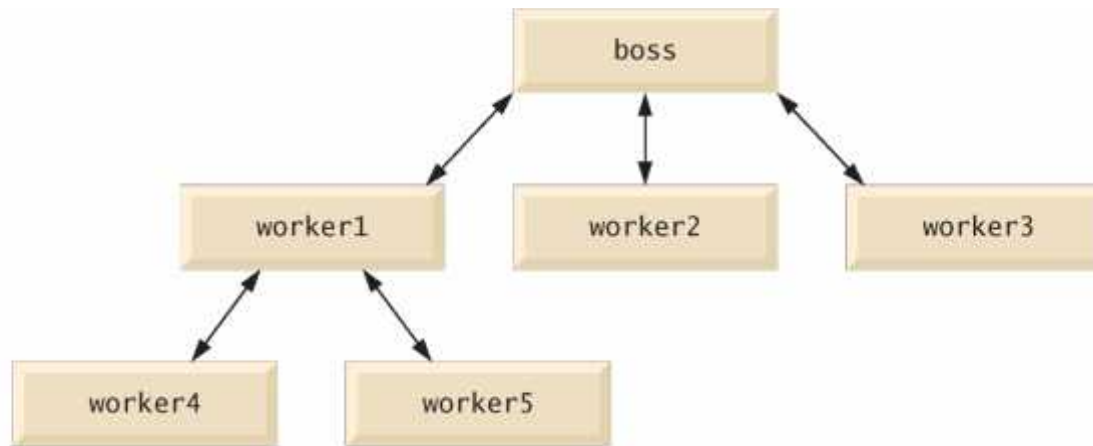


Fig. 5.1 | Hierarchical boss-method/worker-method relationship.

static Methods, static Fields and Class Math

- Sometimes a method performs a task that does not depend on the contents of any object
 - The method applies to the class in which it's declared
 - Known as a **static method** or a **class method**
 - Place the keyword **static** before **the return type** in the declaration
- You can call any **static** method by specifying its class name, followed by a dot (.) and the method name
- *All Math class methods are static*
 - Each is called by preceding the name of the method with the class name **Math** and the dot (.) separator
- Method arguments may be constants, variables or expressions



Software Engineering Observation 5.4

Class `Math` is part of the `java.lang` package, which is implicitly imported by the compiler, so it's not necessary to import class `Math` to use its methods.

Method	Description	Example
<code>abs(x)</code>	absolute value of x	<code>abs(23.7)</code> is 23.7 <code>abs(0.0)</code> is 0.0 <code>abs(-23.7)</code> is 23.7
<code>ceil(x)</code>	rounds x to the smallest integer not less than x	<code>ceil(9.2)</code> is 10.0 <code>ceil(-9.8)</code> is -9.0
<code>cos(x)</code>	trigonometric cosine of x (x in radians)	<code>cos(0.0)</code> is 1.0
<code>exp(x)</code>	exponential method e^x	<code>exp(1.0)</code> is 2.71828 <code>exp(2.0)</code> is 7.38906
<code>floor(x)</code>	rounds x to the largest integer not greater than x	<code>floor(9.2)</code> is 9.0 <code>floor(-9.8)</code> is -10.0
<code>log(x)</code>	natural logarithm of x (base e)	<code>log(Math.E)</code> is 1.0 <code>log(Math.E * Math.E)</code> is 2.0
<code>max(x, y)</code>	larger value of x and y	<code>max(2.3, 12.7)</code> is 12.7 <code>max(-2.3, -12.7)</code> is -2.3
<code>min(x, y)</code>	smaller value of x and y	<code>min(2.3, 12.7)</code> is 2.3 <code>min(-2.3, -12.7)</code> is -12.7

Fig. 5.2 | Math class methods. (Part 1 of 2.)

Method	Description	Example
<code>pow(x, y)</code>	x raised to the power y (i.e., x^y)	<code>pow(2.0, 7.0)</code> is 128.0 <code>pow(9.0, 0.5)</code> is 3.0
<code>sin(x)</code>	trigonometric sine of x (x in radians)	<code>sin(0.0)</code> is 0.0
<code>sqrt(x)</code>	square root of x	<code>sqrt(900.0)</code> is 30.0
<code>tan(x)</code>	trigonometric tangent of x (x in radians)	<code>tan(0.0)</code> is 0.0

Fig. 5.2 | Math class methods. (Part 2 of 2.)

static Methods, static Fields and Class Math (cont.)

- Class `Math` declares commonly used mathematical constants
 - `Math.PI` (3.141592653589793) is the ratio of a circle's circumference to its diameter
 - `Math.E` (2.718281828459045) is the base value for natural logarithms (calculated with `static Math` method `log`)
- These fields are declared in class `Math` with the modifiers `public`, `final` and `static`
 - `public` allows you to use these fields in your own classes
 - `final` indicates a constant—value cannot change
 - `static` allows them to be accessed via the class name `Math` and a dot (`.`) separator
 - `static` fields are also known as `class variables`

static Methods, static Fields and Class Math (cont.)

- Why must `main` be declared `static`?
 - When you execute the Java Virtual Machine (JVM) with the `java` command, the JVM attempts to invoke the `main` method of the class you specify
 - Declaring `main` as `static` allows the JVM to invoke `main` without creating an object of the class

Methods

- In Java, **methods** are used to **create mini programs** within a larger program.
- In general **3 types of methods** are used:
 - A method that **carries out some action**.
 - A method that is **passed parameters** to carry out some action.
 - A method that **carries out some action** and **returns a value**. It may also be passed some parameters.

Type 1 - A Method that carries out some action

Write a program that uses a method to draw stars on the screen.

Solution

We will write an application to solve this problem.

An example of a method that draws 50 stars is as follows:

```
// method to draw stars
public static void drawstars()
{
    for (int j =1 ; j < 50; j = j + 1)
    {
        System.out.print( "*" );
    }
}
```

The method drawstars() is used in the following program:

```
1  // An example of calling a method to draw stars
2
3  public class DrawStarsApp
4  {
5      public static void main( String args[] )
6      {
7
8          // call method to draw stars
9          drawstars();
10
11         // print blank line
12         System.out.println(" ");
13
14         System.out.println("Hello World");
15
16         // call method to drawstars
17         drawstars();
18
19         // print blank line
20         System.out.println(" ");
21     }
22
23
24
25     // method to draw stars
26     public static void drawstars()
27     {
28         for (int j =1 ; j < 50; j = j + 1)
29         {
30             System.out.print("*");
31         }
32     }
33
34 }
```

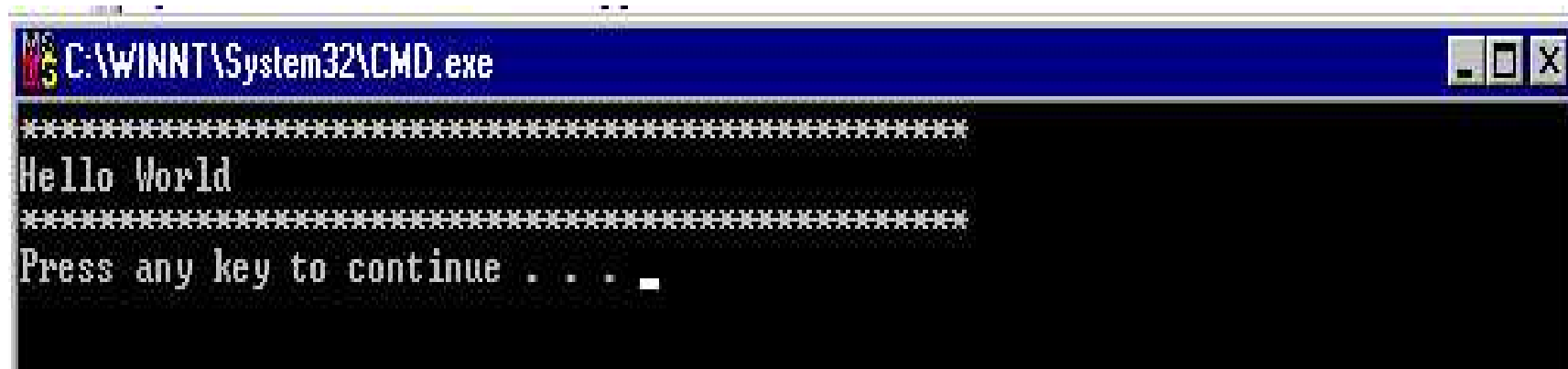
CALLING
METHOD



METHOD
DRAWSTARS



A Sample Testrun



A screenshot of a Windows NT command prompt window. The title bar is blue and contains the text 'C:\WINNT\System32\CMD.exe' and standard window control buttons (minimize, maximize, close). The command prompt has a black background with white text. It displays a series of asterisks, followed by 'Hello World', another series of asterisks, and then 'Press any key to continue . . . ' with a cursor. The asterisks are arranged in a rectangular block, likely representing a simple ASCII art or a decorative separator.

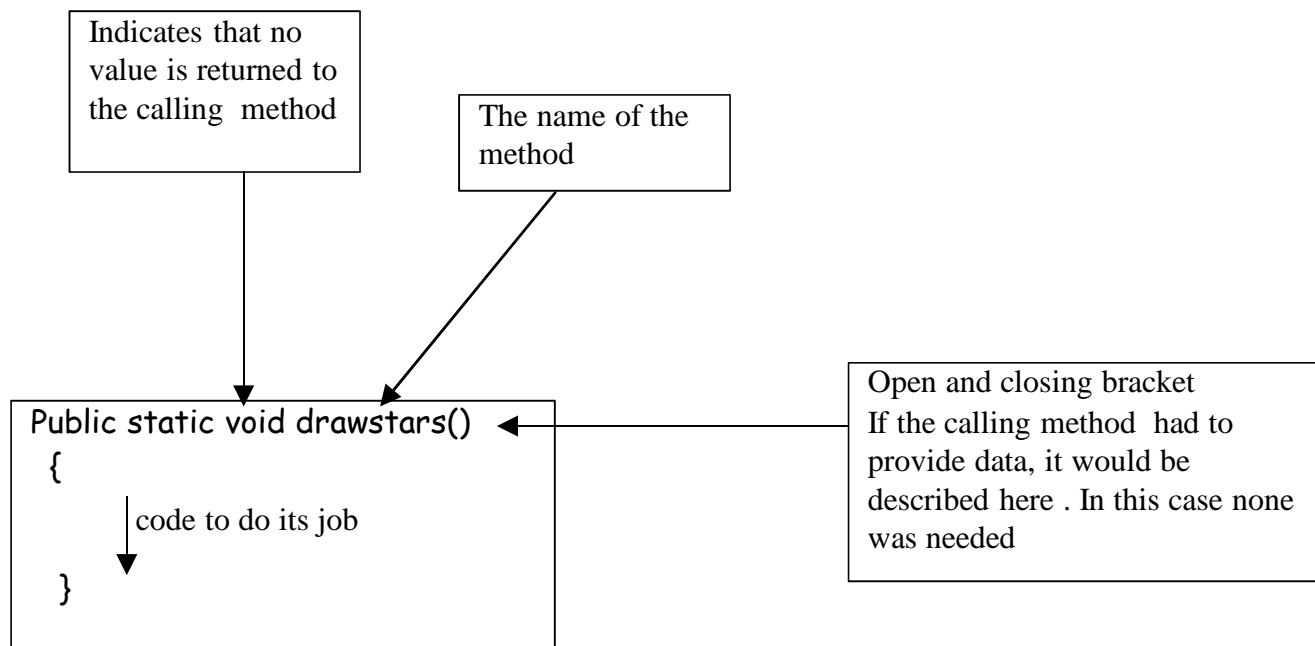
```
C:\WINNT\System32\CMD.exe
*****
Hello World
*****
Press any key to continue . . .
```

Program Notes

- Once method is defined - can be used as often as you like.
- Code for methods should be placed after main method.

The Structure of the Method Called Above

The structure of the method called above



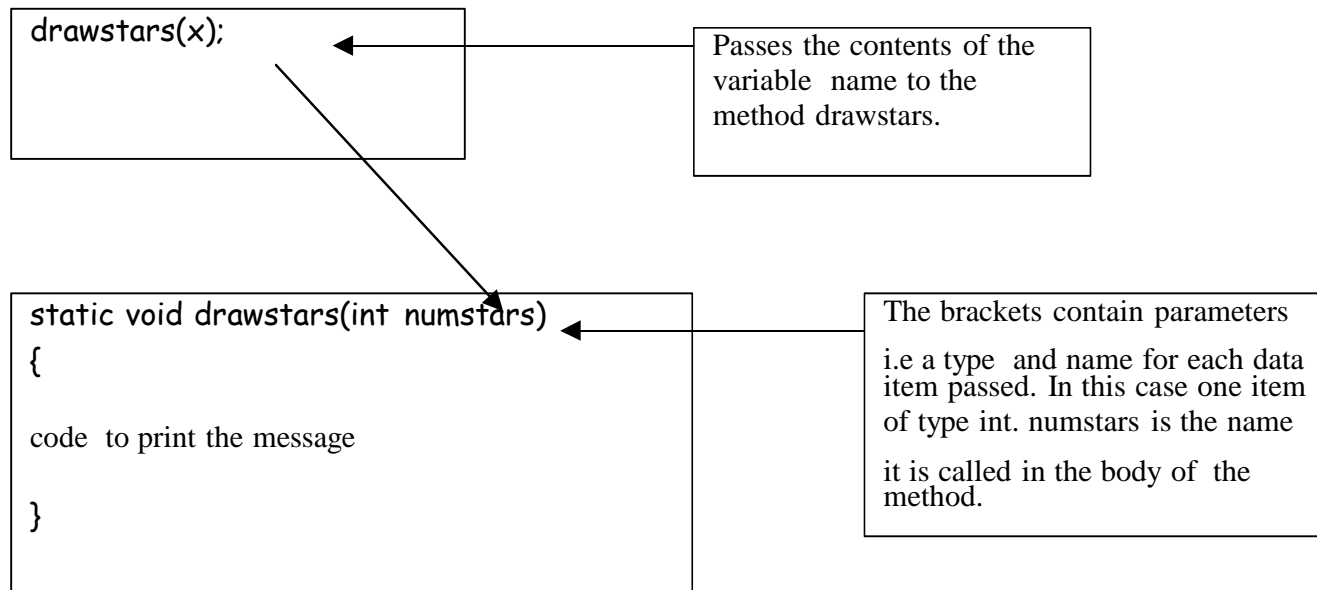
The keyword void indicates that no value is returned to the calling method.

Type 2 - Method passed Parameters to carry out some action

- The method **drawstars()** is limited because it always prints the same number of stars.
- Much better if user could decide on number of stars to be printed.
- To **pass values to methods** when they are invoked, we use what are called **parameters (sometimes**
- **may be called arguments)**.

Type 2 - Method passed Parameters to carry out some action

Format of method that's passed parameters is as follows:



Problem

Modify the drawStarsAppprogram so that a parameter is passed to the drawstars() method telling it how many stars to draw.

Program Solution

```
import java.util.Scanner;
public class DrawStarsParam
{
    public static void main (String[] args)
    {
        Scanner input= new Scanner (System.in);

        int x;

        System.out.println("Please enter number of stars to draw");
        x = input.nextInt();

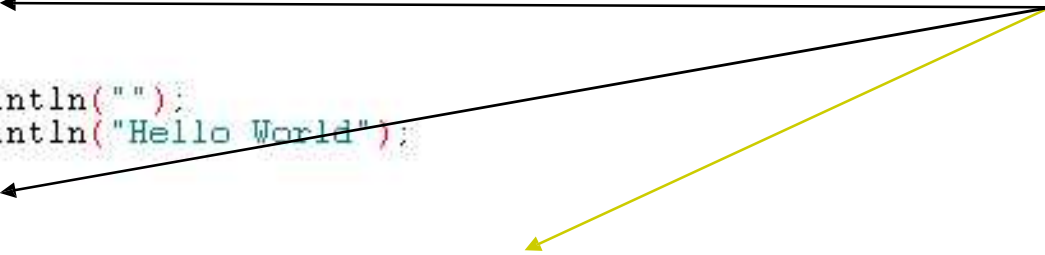
        drawstars(x);

        System.out.println("");
        System.out.println("Hello World");

        drawstars(x);
    }

    public static void drawstars(int numstars)
    {
        (for int j=1; j<=numstars; j=j+1))
        {
            System.out.print("*");
        }
    }
}
```

x is passed
to
numstars



Example Result



```
MS-DOS tp02097a
Auto
Please enter the number of stars to draw
10
*****
Hello World
*****
Press any key to continue . . .
```

Program Notes

Line 11-13

- User asked to enter value.
- 10 was entered and stored in x.

Program Notes (Contd.)

Line 16

```
// call method to drawstars. pass parameter x  
drawstars(x);
```

- This calls the method drawstars and passes the value x, (in this eg 10) to method drawstars.

Line 32-38

```
// method to draw stars  
public static void drawstars(int numstars)  
{  
    for (int j =1 ; j <= numstars; j = j + 1)  
    {  
        System.out.print( "*" );  
    }  
}
```

- This is method **drawstars**.
- The value x (10 in this eg) is passed into variable numstars.
- The for loop then prints out the number of stars required.

Type 3 - A method that carries out some action and returns a value. It may also be passed some parameters.

The basic structure of a method

The type of value to be returned or void if none

Name of the Method

The **parameter list** - the specification of the parameters for the method separated by commas. If the method has no parameters the parentheses are empty.

return_type **methodName(arg1,arg2.....argn)**

```
{  
    local variables  
  
    executable code  
}
```


Declaring Methods

- Class `MaximumFinder` (Fig. 5.3) has two methods—`main` (lines 8–25) and `maximum` (lines 28–41)
- The `maximum` method determines and returns the largest of three `double` values
- Most methods do not get called automatically
 - You must call method `maximum` explicitly to tell it to perform its task

```
1 // Fig. 5.3: MaximumFinder.java
2 // Programmer-declared method maximum with three double parameters.
3 import java.util.Scanner;
4
5 public class MaximumFinder
6 {
7     // application starting point
8     public static void main( String[] args )
9     {
10         // create Scanner for input from command window
11         Scanner input = new Scanner( System.in );
12
13         // prompt for and input three floating-point values
14         System.out.print(
15             "Enter three floating-point values separated by spaces: " );
16         double number1 = input.nextDouble(); // read first double
17         double number2 = input.nextDouble(); // read second double
18         double number3 = input.nextDouble(); // read third double
19
20         // determine the maximum value
21         double result = maximum( number1, number2, number3 );
22     }
```

Fig. 5.3 | Programmer-declared method maximum with three double parameters.
(Part 1 of 3.)

```
23     // display maximum value
24     System.out.println( "Maximum is: " + result );
25 } // end main
26
27 // returns the maximum of its three double parameters
28 public static double maximum( double x, double y, double z )
29 {
30     double maximumValue = x; // assume x is the largest to start
31
32     // determine whether y is greater than maximumValue
33     if ( y > maximumValue )
34         maximumValue = y;
35
36     // determine whether z is greater than maximumValue
37     if ( z > maximumValue )
38         maximumValue = z;
39
40     return maximumValue;
41 } // end method maximum
42 } // end class MaximumFinder
```

Fig. 5.3 | Programmer-declared method `maximum` with three double parameters.
(Part 2 of 3.)

```
Enter three floating-point values separated by spaces: 9.35 2.74 5.1  
Maximum is: 9.35
```

```
Enter three floating-point values separated by spaces: 5.8 12.45 8.32  
Maximum is: 12.45
```

```
Enter three floating-point values separated by spaces: 6.46 4.12 10.54  
Maximum is: 10.54
```

Fig. 5.3 | Programmer-declared method `maximum` with three `double` parameters.
(Part 3 of 3.)

Declaring Methods (cont.)

- A `public` method is “available to the public”
 - Can be called from methods of other classes
- `static` methods in the same class can call each other directly
 - Any other class that uses a static `method` must fully qualify the method name with the class name
- For now, we begin every method declaration with the keywords `public` and `static`
 - You’ll learn about non-`public` and non-`static` methods later.

Declaring Methods (cont.)

- Return type
 - Specifies the type of data a method returns (i.e., gives back) to the calling method after performing its task
- As we saw earlier, in some cases, you'll define methods that perform a task but will not return any information
 - Such methods use the return type `void`.

Declaring Methods (cont.)

- The method name follows the return type
- By convention, method names begin with a lowercase first letter and subsequent words in the name begin with a capital letter (e.g., `nextInt`)
- If the parentheses after the method name are empty, the method does not require additional information to perform its task
- For a method that requires additional information to perform its task, one or more **parameters** represent that additional information
 - Defined in a comma-separated **parameter-list** located in the parentheses that follow the method name
 - Each parameter must specify a type and an identifier
- A method's parameters are considered to be local variables of that method and can be used only in that method's body

Declaring Methods (cont.)

- A method call supplies arguments for each of the method's parameters
 - There must be one argument for each parameter
 - Each argument must be “consistent with” the corresponding parameter's type
- Method header
 - Modifiers, return type, method name and parameters
- Method body
 - Delimited by left and right braces
 - Contains one or more statements that perform the method's task
- A `return` statement returns a value (or just control) to the point in the program from which the method was called

Returning Control from a Method Call

There are three ways to return control to the point at which method was invoked.

These are:

If the method does not return a result then control is returned simply when:

1. the method -ending `}` is reached,
- or
2. by executing the statement

`return;`

Returning Control from a Method Call

If the method does return a result, the statement:

```
return expression;
```

Returns the value of `expression` to the caller.

When a `return` statement is executed, control is returned immediately to the point at which the method was invoked.

Returning Control from a Method Call

The format of a method definition for the 3rd type of method

The format of a method definition is:

```
return-value-type method-name(parameter-list )  
{  
  declarations and statements;  
}
```

Summary of Methods

Type 1 - A Method that carries out some action

Type 2 - Method passed Parameters to carry out some action

Type 3 - A method that carries out some action and returns a value. It may also be passed some parameters.

Reasons for using Methods

Motivations for Methods

There are several motivations for modularising a program with methods.



Motive 1

The “divide-and-conquer” approach makes program development very manageable.

Remember...

*“Take a large problem,
break it into smaller problems,
solve the smaller problems and
thereby solve the big problem”*

Reasons for using Methods



Motive 2

Another motivation is **software re-usability**.

This means using methods as building blocks to create a new program.

With good **method naming and definition**, programs can be created from standardised methods, rather than being built by using customised code.

Reasons for using Methods



Motive 3

The third motivation is to **avoid repeating code chunks** in a program.

Packaging code as a method allows that code to be executed from several locations in a program simply by calling the method .

Methods

Ideally...

- Each method should be limited to performing a single, well-defined task ,

and

- the method name should express that task.

Java API Packages

- Java contains many predefined classes that are grouped into categories of related classes called packages
- Known as the Java Application Programming Interface (Java API), or the Java class library
- Overview of the packages in Java SE 6
 - <http://java.sun.com/javase/6/docs/api/overview-summary.html>
- Additional information about a predefined Java class's methods
 - <http://java.sun.com/javase/6/docs/api/>
 - **Index** link shows alphabetical list of all the classes and methods in the Java API
 - Locate the class name and click its link to see the online description of the class
 - **METHOD** link shows a table of the class's methods
 - Each **static** method will be listed with the word “**static**” preceding the method's return type

Package	Description
<code>java.applet</code>	The Java Applet Package contains a class and several interfaces for creating Java applets—programs that execute in web browsers. Applets are discussed in Chapter 23, Applets and Java Web Start; interfaces are discussed in Chapter 10, Object-Oriented Programming: Polymorphism.)
<code>java.awt</code>	The Java Abstract Window Toolkit Package contains the classes and interfaces required to create and manipulate GUIs in early versions of Java. In current versions of Java, the Swing GUI components of the <code>javax.swing</code> packages are typically used instead. (Some elements of the <code>java.awt</code> package are discussed in Chapter 14, GUI Components: Part 1, Chapter 15, Graphics and Java 2D™, and Chapter 25, GUI Components: Part 2.)
<code>java.awt.event</code>	The Java Abstract Window Toolkit Event Package contains classes and interfaces that enable event handling for GUI components in both the <code>java.awt</code> and <code>javax.swing</code> packages. (See Chapter 14, GUI Components: Part 1 and Chapter 25, GUI Components: Part 2.)

Fig. 5.5 | Java API packages (a subset). (Part 1 of 4.)

Package	Description
<code>java.awt.geom</code>	The Java 2D Shapes Package contains classes and interfaces for working with Java's advanced two-dimensional graphics capabilities. (See Chapter 15, Graphics and Java 2D™.)
<code>java.io</code>	The Java Input/Output Package contains classes and interfaces that enable programs to input and output data. (See Chapter 17, Files, Streams and Object Serialization.)
<code>java.lang</code>	The Java Language Package contains classes and interfaces (discussed bookwide) that are required by many Java programs. This package is imported by the compiler into all programs.
<code>java.net</code>	The Java Networking Package contains classes and interfaces that enable programs to communicate via computer networks like the Internet. (See Chapter 27, Networking.)
<code>java.sql</code>	The JDBC Package contains classes and interfaces for working with databases. (See Chapter 28, Accessing Databases with JDBC.)

Fig. 5.5 | Java API packages (a subset). (Part 2 of 4.)

Package	Description
<code>java.text</code>	The Java Text Package contains classes and interfaces that enable programs to manipulate numbers, dates, characters and strings. The package provides internationalization capabilities that enable a program to be customized to locales (e.g., a program may display strings in different languages, based on the user's country).
<code>java.util</code>	The Java Utilities Package contains utility classes and interfaces that enable such actions as date and time manipulations, random-number processing (class <code>Random</code> , Section 5.9) and the storing and processing of large amounts of data.
<code>java.util.concurrent</code>	The Java Concurrency Package contains utility classes and interfaces for implementing programs that can perform multiple tasks in parallel. (See Chapter 26, Multithreading.)
<code>javax.media</code>	The Java Media Framework Package contains classes and interfaces for working with Java's multimedia capabilities. (See Chapter 24, Multimedia: Applets and Applications.)

Fig. 5.5 | Java API packages (a subset). (Part 3 of 4.)

Package	Description
<code>javax.swing</code>	The Java Swing GUI Components Package contains classes and interfaces for Java's Swing GUI components that provide support for portable GUIs. (See Chapter 14, GUI Components: Part 1 and Chapter 25, GUI Components: Part 2.)
<code>javax.swing.event</code>	The Java Swing Event Package contains classes and interfaces that enable event handling (e.g., responding to button clicks) for GUI components in package <code>javax.swing</code> . (See Chapter 14, GUI Components: Part 1 and Chapter 25, GUI Components: Part 2.)
<code>javax.xml.ws</code>	The JAX-WS Package contains classes and interfaces for working with web services in Java. (See Chapter 31, Web Services.)

Fig. 5.5 | Java API packages (a subset). (Part 4 of 4.)