### Topic 2.3 - Methods and the Java API

|  |
| --- |
| Methods in Java  A Java method is a collection of statements that are grouped together to perform an operation.  **public static int methodName (int a, int b) {**  **//body**  **}**  **public static -> modifiers**  **int-> return type**  **methodName -> name of the method**  **a,b -> formal parameters**  **int a, int b -> list of parameters**  Method definition consist of a method header and a method body  **modifier returnType nameOfMethod(Parameter List) {**  **// method body**  **}**  35 |

|  |
| --- |
| Methods in Java (cont…)  **modifier:** defines the access type of the method and it is optional to use.  **returnType:** a value that may be returned by a method.    **nameOfMethod:** This is the method name. The method name and the parameter list form the *method signature.*  **Parameter List:** The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain *zero* parameters.  **method body:** The method body defines what the method does with statements.  36 |

|  |
| --- |
| Why use methods?   * It is common (good) practice to break up the implementation of a program into different methods, where each method performs a specific task that the larger program requires. * Methods often require some information to be supplied in order to perform the required task and can optionally return a result to the caller. * Any methods that you define must be part of a class – they cannot exist on their own. * Many common tasks have been implemented as methods in the Java API classes!   37 |

|  |
| --- |
| Method implementation   * The formal parameters in a method are essentially variables containing information that the caller has supplied. * You can (and should) declare local variables within a method if it needs to store or “remember” values as part of the task it is performing. * If the method has a non-void return type then it must return a literal value or variable of the corresponding type.   **public int multiply (int a, int b){**  **int product; // declaring local variable**  **// code to perform required task**  **product = a \* b;**  **// return statement needed here**  **return product;**  **}**  38 |

|  |
| --- |
| Calling methods  When creating a method, one defines what a method is to do. To use a method to perform its function, one has to call or invoke it.  There are two ways to call a method; the choice is based on whether the method returns a **value** or not and whether the method you call is in the **same class** or not ( *to be discussed later*)  When a program calls a method, program control is transferred to the called method. A called method returns control to the caller when its return statement is executed or when its method‐ending closing brace is reached.   * If the method returns a value, a call to the method is usually treated as a value. * If the method returns void, a call to the method must be a statement.   39 |

|  |
| --- |
| An example to demonstrate how to define a method and how to call it:  **class MethodDemo1 {**  **public static void main(String [] args){**  **int x = 5, y = 2;**  **int result;**  **result = multiply(x, y); // method call System.out.println(x + “ \* “ + y + “ = “ + result);**  **}**  **// method must be static because main() is static public static int multiply (int a, int b) {**  **int product;**  **product = a \* b;**  **return product; // return value to caller**  **}**  **}**  40 |

|  |
| --- |
| Parameter passing / return values  * In the previous example the variables ‘a’ and ‘b’ were passed along as actual parameters to the method call. * For primitive type parameters a copy of the value stored in the each actual parameter is passed along to the corresponding formal parameter within the method. * A copy of the value being returned is sent back to be stored in the corresponding variable   **result = multiply(x, y);**  41 |

|  |
| --- |
| More on return values  * It is possible to ignore the value being returned by a method.   **class MethodDemo2 {**  **public static void main(String [] args) {**  **int x = 5, y = 2;**  **multiply(x, y); // return value is ignored**  **}**  **// method must be static because main() is static public static**  **int multiply (int a, int b) {**  **int product;**  **product = a \* b;**  **return product;**  **}**  **}**  42 |

|  |
| --- |
| Methods without return values  * It is also possible to have methods which do not return a value to the caller by specifying the return type as ‘**void**’.   **class MethodDemo3 {**  **public static void main(String [] args) {**  **int x = 5, y = 2;**  **multiply(x, y); // no return value**  **}**  **// use void for the return type if the method**  **// does not need to return a value**  **public static void multiply (int a, int b) {**  **int product;**  **product = a \* b;**  **System.out.println(a + “ \* “ + b + “ = “ + product);**  **}**  **}**  43 |

|  |
| --- |
| Passing Parameter by Value **public class swappingExample {**  **public static void main(String[] args) {**  **int x = 30; int y = 45;**  **System.out.println("Before swapping, x = " + x + "**  **and y = " + y);**  **// Invoke the swap method**  **swapFunction(x, y);**  **System.out.println("\n Now, Before and After swapping values**  **will be same here :");**  **System.out.println("After swapping, x = " + x + "**  **and y is " + y);**  **}**  **public static void swapFunction(int x, int y) {**  **System.out.println("Before swapping(Inside), x = " + x + "**  **y= " + y);**  **// Swap x with y**  **int z = x;**  **x = y;**  **y = z;**  **System.out.println("After swapping(Inside), a = " + a + "**  **b = " + b);**  **}**  **}** 44 |

|  |  |
| --- | --- |
| |  | | --- | | * When a method is called, the actual parameter values are computed and passed into the formal parameter variables. * Thus, when a primitive is passed, any changes made to the formal parameter (the copy) will not affect the actual parameter (original) in the calling method. * However, when a reference to an object is passed, the formal parameters in the called method can be used to directly manipulate the original object. |  Primitives & Object references asarguments 45 |

|  |
| --- |
| Sample Program: Passing Primitives & Object references **class MyInt {**  **public MyInt(int n) { val = n; }**  **public void add(int n) { val += n; }**  **public int getVal() { return val; }**  **private int val;**  **}**  **public class PrimsAndObjects {**  **public static void main (String[] args) {**  **int num1 = 10;**  **MyInt num2 = new MyInt(10);**  **addOne(num1, num2);**  **System.out.println("primitive num1 = " + num1+**  **" object referred by num2 = " + num2.getVal());**  **}**  **public static void addOne(int x, MyInt y) {**  **x += 1;**  **y.add(1);**  **}**  **}**  46 |