**Checkpoint Answers Chapter 6**

**6.1**

Some of the benefits of using methods are:

* Faster development time by reuse of code
* Reduces complexity of the code
* Makes the code easier to maintain

**6.2**

A number of properties about a given method need to be decided before it can be defined:

* What modifiers are suitable
* The type of the value, if any, that shall be returned
* The name of the method
* What parameters are needed as data for the method to be able to do its work
* What the method is intended to do and how it shall do it

When all of the above decisions have been made so can the method be defined. An example of a method definition looks like this:

public static int max(int num1, int num2) {

if (num1 > num2)

return num1;

else

return num2;

}

The keywords public and static are examples of modifiers. The following int indicates that the method will return an int after being executed. The name of the method is max and it takes two int's as data parameters.

The part between the braces is the body of the loop and it this part that does the actual work. In this case so will it be checked which of the parameters that is biggest and the value of this parameter is returned to the caller of the method.

A method is invoked by using the name of the method combined with the same number and type of actual parameter values as in the definition of the method.

An invocation of the max method can for example look like this:

int biggestNumber = max(15, 42);

**6.3**

The body of the max method can be shortened by the use of the conditional operator.

return ( num1 > num2 ) ? num1 : num2;

**6.4**

A call to a method with a void return type is always a statement itself. A call to a value-returning method can also be a statement by itself.

**6.5**

The return type of the main method is void.

**6.6**

Omitting a return statement in a value-returning method will cause a syntax error.

It is possible to have a return statement in a void function, this will end the execution of the method.

It is a syntax error to have a void function return a value.

**6.7**

The term parameter is used to refer to the list of variables in a method declaration.

The term argument refers to the actual values that are passed in when the method is invoked.

A method signature is made up of the name and the parameter list of a given method. Note that the return type is not part of the method signature.

**6.8**

**(a)**

public static double getCommission(double SalesAmount, double commissionRate)

**(b)**

public static void displayCalendar(int mont, int year)

**(c)**

public static double squareRoot(double num)

**(d)**

public static boolean isEven(int num)

**(e)**

public static void displayMessage(String message, int count)

**(f)**

public static double monthlyPayment(double loanAmount, int years, double interestRate)

**(g)**

public static char toUpperCase(char letter)

**6.9**

public class Test {

public static method1(int n, m) {

n += m;

method2(3.4);

}

public static int method2(int n) {

if (n > 0)

return 1;

else if (n == 0)

return 0;

else if (n < 0)

return -1;

}

}

There are multiple errors in the above code.

The return type for method1 is missing, it shall be void.

The type of the second parameter in method1 is missing, it shall most likely be of type int.

The second method, named method2, is in method1 invoked with a double argument but the parameter of method2 is of typ int. This causes an syntax error, this can be fixed by changing the method signature of method2.

The compiler will think that it is possible for method2 to not return any value in some situations. Remove the last if to make the code compile without changing the behavior.

A fully corrected code look like this:

public class Test {

public static void method1(int n, int m) {

n += m;

method2(3.4);

}

public static int method2(double n) {

if (n > 0)

return 1;

else if (n == 0)

return 0;

else

return -1;

}

}

**6.10**

public class Test {

public static double method(double i, double j) {

while (i < j) {

j--;

}

return j;

}

}

**6.11**

The arguments passed to a method must be passed in the same order as in the method signature and the types of all argument must be compatible to the formal parameters in the method signature.

An argument can have the same name as a formal parameter, this is for example a valid program:

public class CheckPoint\_06\_11 {

public static void main(String[] args) {

int i = 5;

int j = 10;

printSum(i, j);

}

public static void printSum(int i, int j) {

System.out.println(i + j);

}

}

**6.12**

There is a mistake when calling the method. The order of the two parameters have been mixed up.

Another mistake is in the the body of nPrintln where n is declared even though n is used as an name for one of the formal parameters.

**6.13**

By definition, pass by value means you are making a copy in memory of the actual parameter's value that is passed in. All work done in the method is then done on this copy.

**(a)**  
Output is:

0

It is only the local copy that is altered by the method, the original is left unchanged.

**(b)**  
Output is:

2

2 4

2 4 8

2 4 8 16

2 4 8 16 32

2 4 8 16 32 64

**(c)**  
Output is:

Before the call, variable times is 3

n = 3

Welcome to Java!

n = 2

Welcome to Java!

n = 1

Welcome to Java!

After the call, variable times is 3

What happens is that the value of the variable times is copied to n. The value of n is then changed but the value of times stays unchanged.

**(d)**  
Output is:

1

2 1

2 1

4 2 1

i is 5

**6.14**

Stack content just before the max-method is invoked:

| |

| max 0 |

-------------

Stack content just entering the max-method, we use more stack now because max need to store some data:

| |

| max 0 |

| value2 2 |

| value1 1 |

-------------

| max 0 |

-------------

Stack content just before return from the max-method, that now have done some writing in the memory:

| |

| max 0 |

| value2 2 |

| value1 1 |

-------------

| max 0 |

-------------

Stack content after return from the max-method, the memory needed by the max-method is now free again:

| |

| max 0 |

-------------

**6.15**

Method overloading in Java occurs when two or more methods in the same class have the exact same name but different parameters.

It is allowed to have identical method names, as long as the parameter types are sufficintly different.

It is not allowed to have overloaded methods where the difference is just based on return types or modifiers.

**6.16**

Both methods defined in this check point have the exact same signature, this is not allowed since there is no way to know which of the methods to invoke.

**6.17**

**(a)**  
The second method will be invoked in this case.

**(b)**  
The second method will be invoked in this case.

**(c)**  
The first method will be invoked in this case.

**6.18**

A local variable in Java is a variable that is declared within the body of a method. This variable can be used variable only within that method. Other methods in the class are not even aware that the variable exists.

**6.19**

The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.