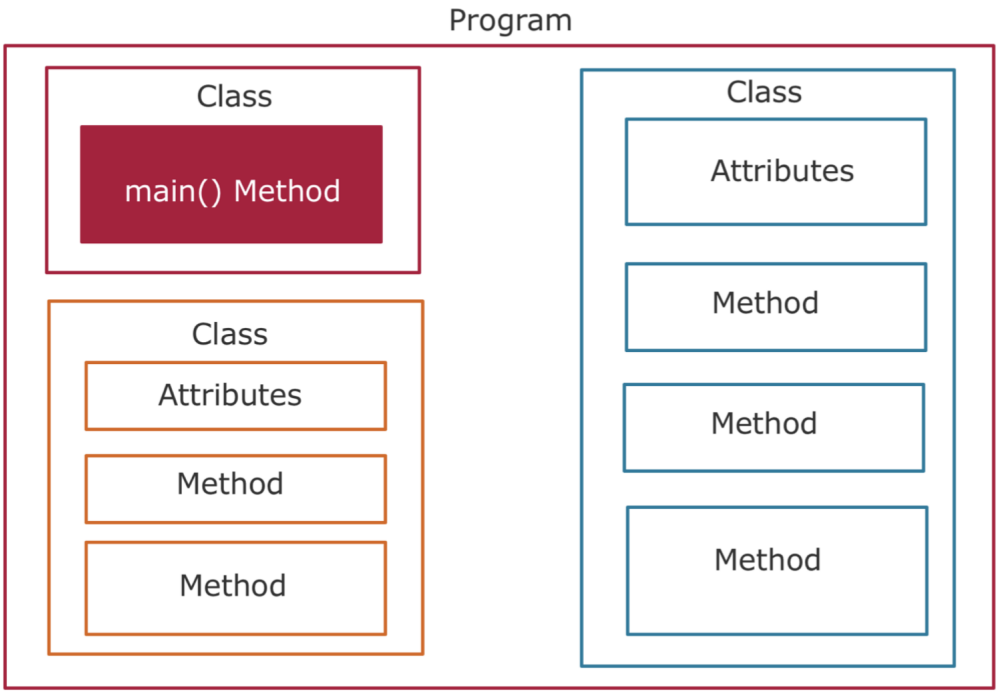
Summary: Object-Oriented Programming in Java on openSAP

1 Java Basics

1.1 Program Structure

Each Java program consists of at least one class. Each Java class[1] is defined inside of a file. The file name extensions should be *.java*. The class identifier and the file name must match. Inside classes, Methods and Attributes are defined.



1.1.1 main() method

Each Java program starts in the main() method. The main() method is defined in one of the classes of the program. The main() method can theoretically be defined in any class. Practically it is often defined in a separate class that does nothing else.

```
public static void main(String[] args){
   //every Java program starts in the main method
}
```

[1] except inner classes: https://docs.oracle.com/javase/tutorial/java/javaOO/innerclasses.html

1.2 Syntax and Code Elements

Java is case sensitive. That means that *MyIdentifier* and *Myidentifier* are two different identifiers.

1.2.1 Statements

Statements end with a semicolon.

1.2.2 Expressions

An expression is everything that can be resolved to a value, e.g. a variable, a calculation or a method call.

1.2.3 Code Conventions

- Class identifiers begin with a capital letter
- Method, attribute and variable identifiers start with a lowercase letter
- In order to use descriptive identifiers, we can combine words (e.g. MyIdentifier). For combined words CamelCase is used
- Methods that return a boolean are prefixed with *is*, e.g. isHuman

1.2.4 Comments

Single line comments are started with two slashes.

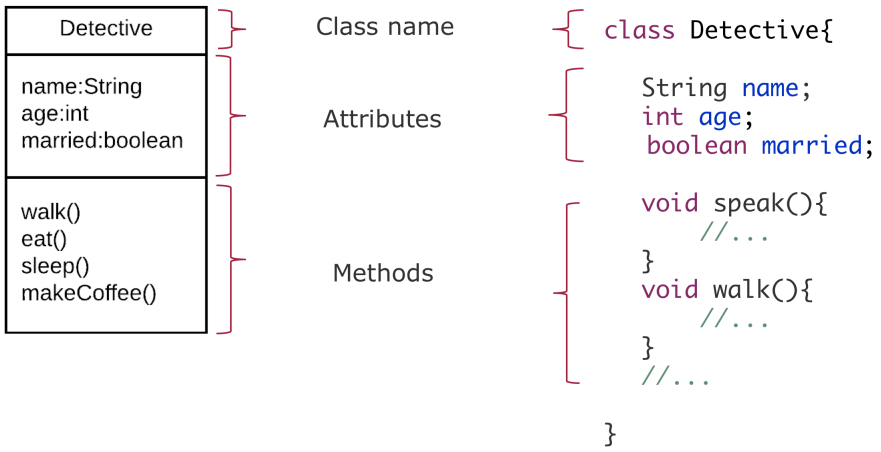
```
//this is a single line comment
```

Multi-line comments start with /* and end with */.

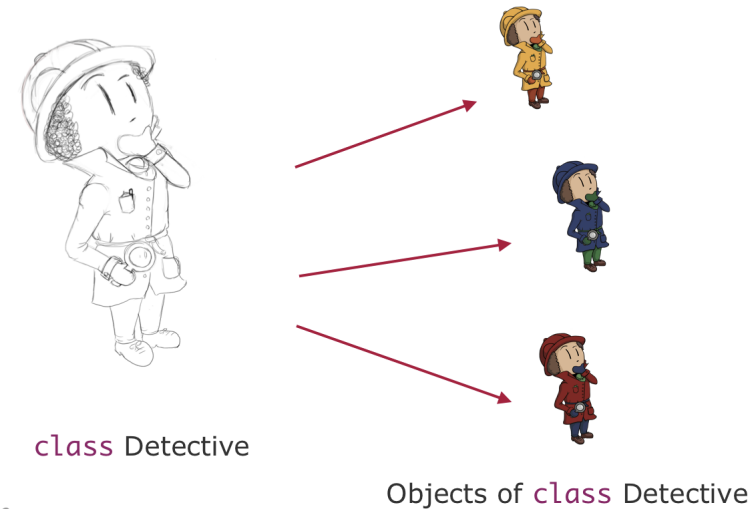
```
/*
 * This is a
 * multi-line comment
 */
```

2 Classes and Objects

Java is an object-oriented programming language. In Java, the basic program unit is a class. A program can consist of many classes. Classes define attributes and methods.



Classes do not represent a tangible entity. They serve as a blueprint for one or many objects. Objects are concrete representations that have unique values. Out of one class we can instantiate (create) many objects. These objects can differ in their state and behavior.



2.0.1 Object Instantiation

Objects are instantiated with the new operator:

```
//<<class identifier>> <<object identifier>> = new <<class identifier>>();
Detective ourDetective = new Detective();
```

2.1 State

The things an object knows or has are also called the state of an object. The state is defined by attributes.

```
class Car{
   //convention: attributes are declared at the top of the class body
   String model;
}
```

2.2 Behavior

The behavior of an object is defined by its methods. The things an object can do are the behavior of an object.

```
class Car{

   //methods are defined below attributes (if the class has any)
   void drive(){
      // ...
   }
}
```

2.3 State and Behavior

The state of an object influences its behavior. The behavior of an object influences its state.
If we consider the class Car, we can imagine that the amount of fuel influences whether a car can drive or not (state influences behavior). Additionally, a method refuel could influence the amount of fuel (behavior influences state).

```
class Car{

double fuelInLiters;

   void drive(){
      // check for fuel first
      // ...
   }
   void refuel(){
      //...
   }
}
```

3 Data Types and Operators

Variables as well as attributes are containers for data. They must have a unique identifier and a data type.
The difference between an attribute and a variable is that an attribute belongs to an object, whereas a (local) variable is only valid within its scope (e.g. method).

3.1 Terminology

Term	Description
Declare a variable	Set an identifier for the variable (and allocate memory for it)
Assign a value	Set the value of the variable
Initialize a variable	First assignment of a value to a variable
Define a variable	Initialization and declaration of a variable in one step

Here are examples with the data type String:

```
//Declare a variable: <<data type>> <<identifier>>;
String example;

//Initialize a variable: <<identifier>> = <<value>>;
example = "Hello!";

//Reassign a variable value: <<identifier>> = <value>>;
example = "Hello openHPI!";

//Define a variable: <<data type>> <<identifier>> = <<value>>;
String anotherExample = "Hello World!";
```

3.2 Data Types

When defining or declaring a variable / an attribute, we use data types. Data types can either be primitive or complex.

3.2.1 Primitive Data Types

Data Type	Description
boolean	Represent truth values, can be true or false
char	Single Characters
byte	Integers ranging from -128 to 127
short	Integers ranging from -32768 to 32767
int	Integers ranging from -2147483648 to 2147483647
long	Integers ranging from -9223372036854775808 to 9223372036854775807
float	Floating Point Numbers ranging from -3.4028235E38 to 3.4028235E38
double	Floating Point Numbers ranging from -1.7976931348623157E308 to 1.7976931348623157E308

3.2.2 Complex Data Types

Besides primitive data types, every Java class can be used as a data type. Many of these classes are defined in the Java API (e.g. String, used for text). Moreover, we can define classes ourself and use them as data types (e.g. class Detective, Car).

3.3 Arithmetic Operators

Calculations on ints and doubles can be done with the standard set of arithmetic operators.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division (int: Integer part of the division / pre-decimal point position) (double: normal division)
%	Modulo (Remainder of the division)

3.3.1 Shorthand Forms

Shorthand Forms are often used as they are slightly shorter than the verbose form.

Operator	Description
+=	Adds right operand to the left operand, assigns the result to left operand
-=	Subtracts right operand from the left operand, assigns the result to left operand
*=	Multiplies right operand with the left operand, assigns the result to left operand
/=	Divides left operand by the right operand, assigns the result to left operand
++	Increment by 1
- -	Decrement by 1

Here is an example how they are used in code:

```
int shorthand = 0;
int nonShorthand = 0;

shorthand +=3;
nonShorthand = nonShorthand + 3;

shorthand -=2;
nonShorthand = nonShorthand - 2;

shorthand *= 6;
nonShorthand = nonShorthand * 6;

shorthand /= 2;
nonShorthand = nonShorthand / 2;

shorthand++;
nonShorthand = nonShorthand + 1;

shorthand--;
nonShorthand = nonShorthand - 1;
```

3.4 Comparison Operators

Comparison operators are used to test for equality and inequality of values. The following operators should be used on primitive data types.

Operator	Description
==	Equal
!=	Not equal

The following comparison operators are mostly used on numerical values. The result of the comparison is a boolean value (true or false).

Operator	Description
>	Greater than
<	Smaller than
>=	Greater or equal
<=	Smaller or equal

3.5 equals()

The operators == and != are used on primitive data types. If we want to test whether two complex data types (e.g. String) are equal, we should use the equals() method.

```
//Compare object data types:
//<<object1 identifier>>.equals(<<object2 identifier>>);
String first    = "Hello";
String second   = new String("Hello");
boolean doNotDoThis = (first == second);
boolean doThat = first.equals(second);
```

3.6 Logical Operators

Logical Operators can be used on *booleans*:

Operator	Description	Evaluates to true
&&	AND	if both expressions evaluate to true
||	OR	if at least one expression is true
!	NOT (Negation)	if expression is false (flips logical values)

4 Methods and Attributes

4.1 Methods

Methods specify functionality for all objects of a class. Method definitions are always placed within classes. Syntax for defining a method:

```
<<return data type>> <<methodIdentifier>>(){
   //statements here
}
```

Methods can be called within the same class. For this, we simplyuse the method identifier. If we want to call a method on an object of another class, we use the obejct identifier and the dot operator, followed by the method identifier.

```
//method call within the same class:
<<method identifier>>();

//Method Call on an object (with dot-operator):
<<object identifier>>.<<method identifier>>();
```

4.1.1 Return type and value

Each method has a return type. If the method does not return a value, the return type *void* is used.
If the method does return a value, any data type (primitive as well as complex) can be used.

```
//a method that does not return a value
void myMethod(){
   System.out.println("Hello World");
}

//a method that returns a String
String myMethodWithReturnValue(){
   return "Hello World!";
}
```

The return value can either be stored by the caller in a variable for further use or the value can be ignored by the caller.

```
void anotherMethod(){
   //store return value
   String output = myMethodWithReturnValue();

   //ignore return value
   myMethodWithReturnValue();
}

String myMethodWithReturnValue(){
   return "Hello World!";
}
```

4.2 Attributes

Attributes describe characteristics and properties of objects.
Attribute definitions are always placed within classes. Attributes can either be defined or declared inside of the class body, but outside of methods. Initializing or (re-)assigning an attribute has to take place within methods.
All instances of the class will have these attributes with unique values.

```
class Detective{
   String name; //declaration : <<data type>> <<identifier>>;
   int age = 42; //definition : <<data type>> <<identifier>> = <<value>>;
}
```

If an attribute is defined with an initial value, this value will be set initially for all objects that are instantiated. This value can be changed later on for every individual object.

4.2.1 Dot Operator

The dot operator can be used to access objects' attributes:

```
Detective duke = new Detective();
//Retrieve Attribute Value: <<object identifier>>.<<attribute identifier>>;
String name = duke.name;

//Set Attribute Value: <<object identifier>>.<<attribute identifier>> =
    <<value>>;
duke.name = "Sir Duke";
```

4.3 Assignment and Comparisons

Operator	Effect
=	**Assignment** declaring, defining, or assigning values to Variables or Attributes)
==	**Comparison** of primitive data types (e.g. int, double, boolean, ...)
equals()	**Comparison** of object data types (e.g. String, Detective, ...)

5 Control Structures

In Java there are different control structures. Conditions allow you to execute code only under certain conditions. Loops allow you to repeatedly execute code, depending on a condition.

5.1 Conditions: if / else

To execute code only if a certain condition holds, we need conditional (boolean) expressions combined with the if and else keyword.

```
if (<<conditional expression>>) {
   //do this
} else {
   //do something different
   //else part ist optional
}
```

The if code block is only executed if the conditional expression evaluates to true.
The *else* code block ist the negation of the first conditional expression. Therefore, no condition is placed after else. The *else* code block is executed if the first conditional expression evaluates to false.

If necesary, we can also test for more expressions, by adding (one to many optional) *else if* blocks. An else if block is only executed if all of the prior expressions are false and its own expression is true.

```
if (<<conditional expression>>) {
   //do this
} else if (<<other conditional expression>>){
   // can have multiple else if code blocks
} else {
   //executed if all prior expressions evaluate to false
}
```