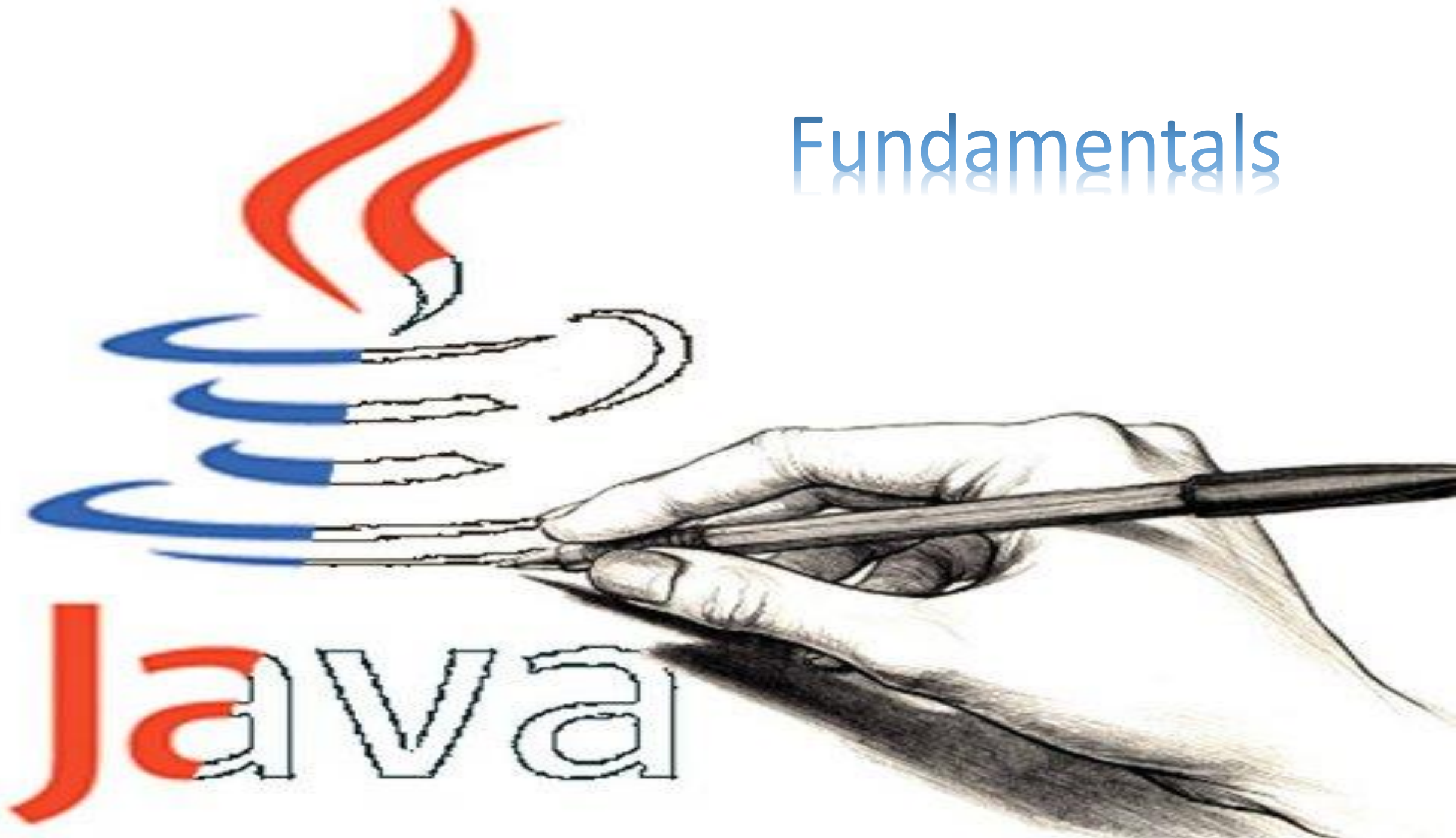
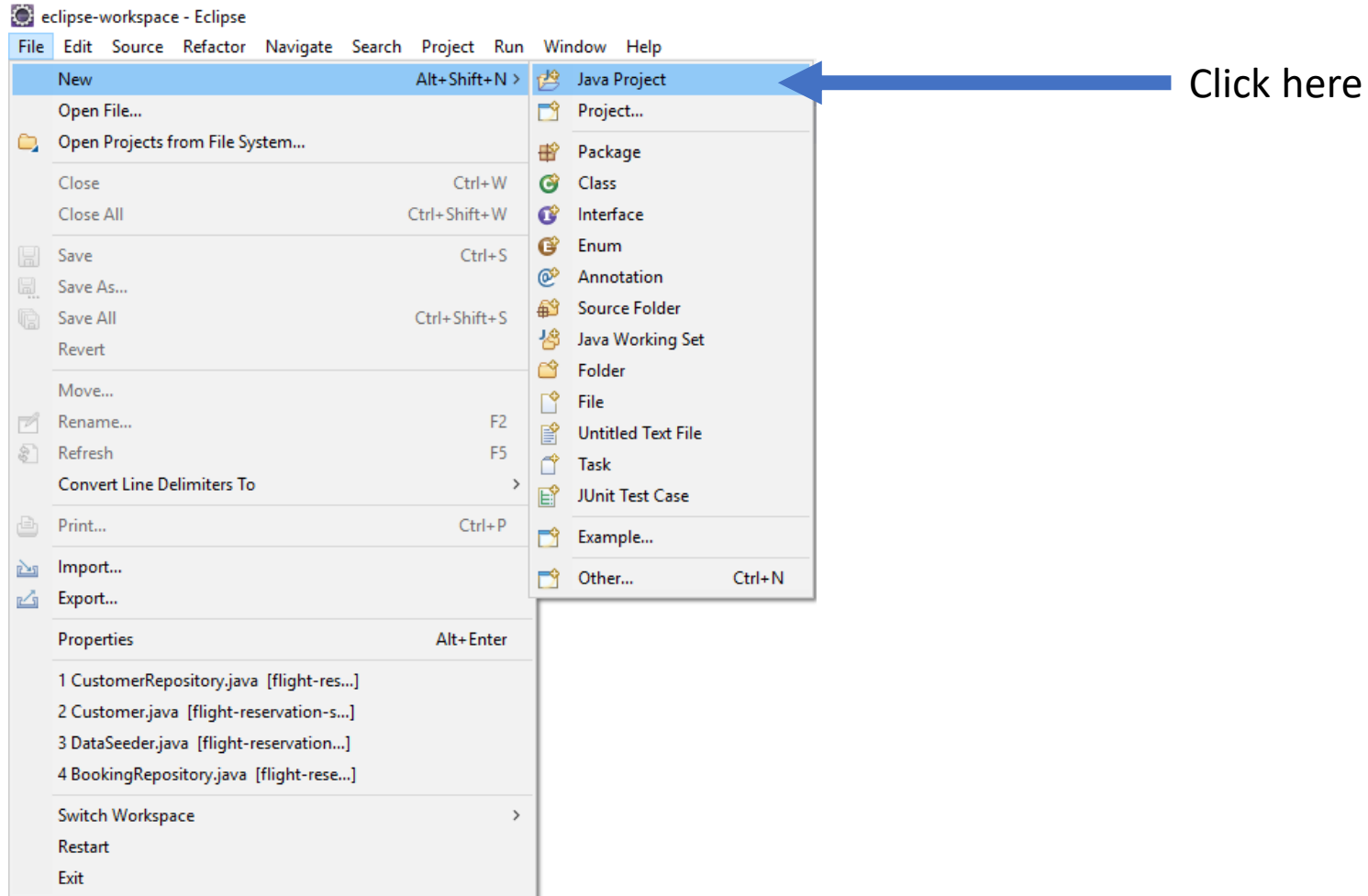


Fundamentals



Creating a Java Project in Eclipse



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: HelloWorld

☒ Use default location

Location: C:\Users\Konsult\eclipse-workspace\HelloWorld

JRE

☒ Use an execution environment JRE: JavaSE-1.8

☐ Use a project specific JRE: jdk1.8.0_151

☐ Use default JRE (currently 'jdk1.8.0_151')

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files

Working sets

☐ Add project to working sets

Working sets:

The default compiler compliance level for the current workspace is 9. The new project will use a project specific compiler compliance level of 1.8.

< Back Next > Finish

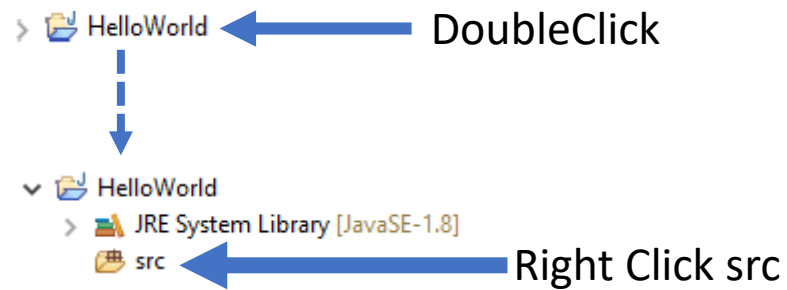
Here you give your project a name, by convention it should start with capital letter.

"Upper camel case is a naming convention in which several words are joined together, and the first letter in each word is capitalized."

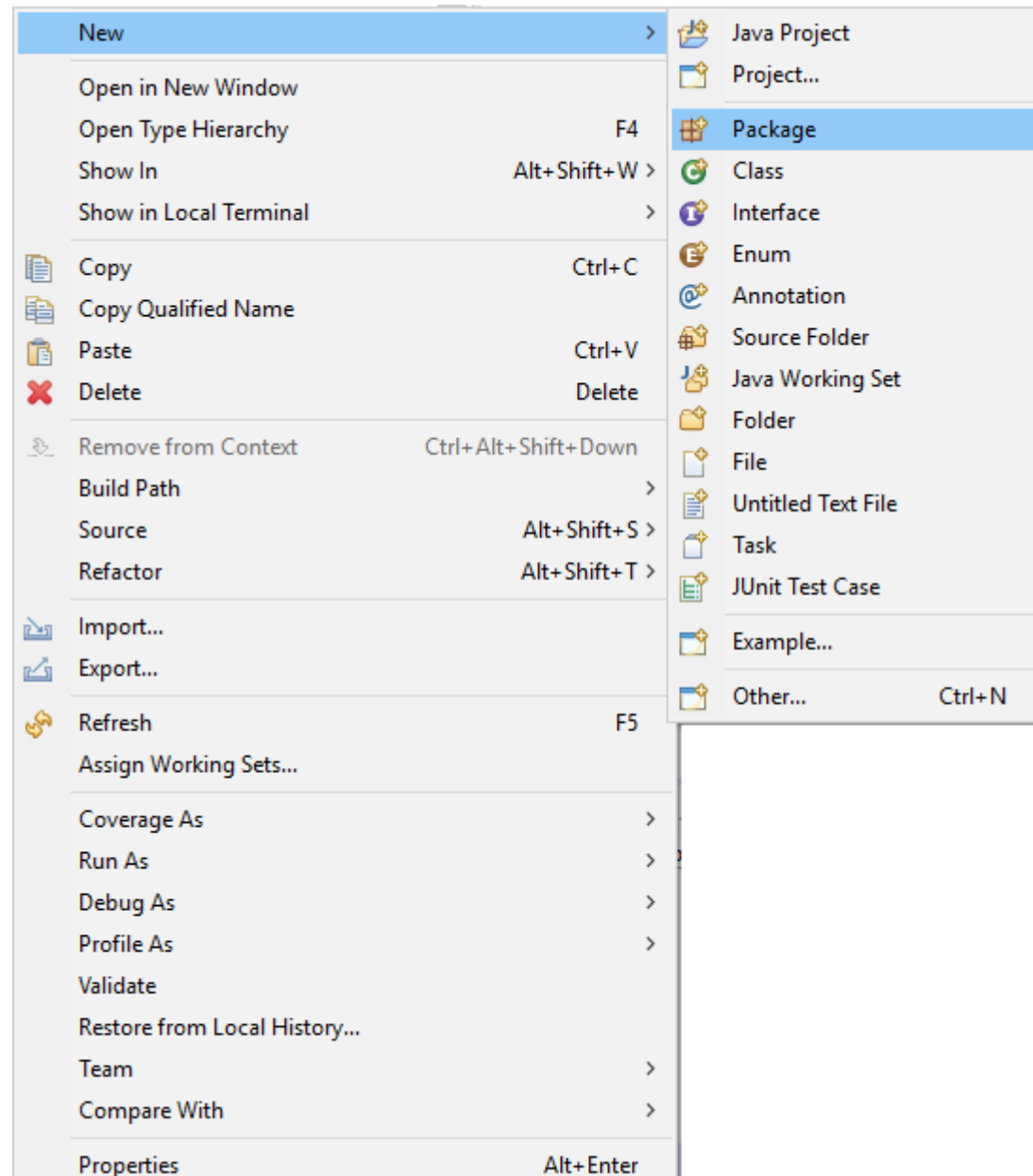
Project names should by convention be Upper camel case in Java.

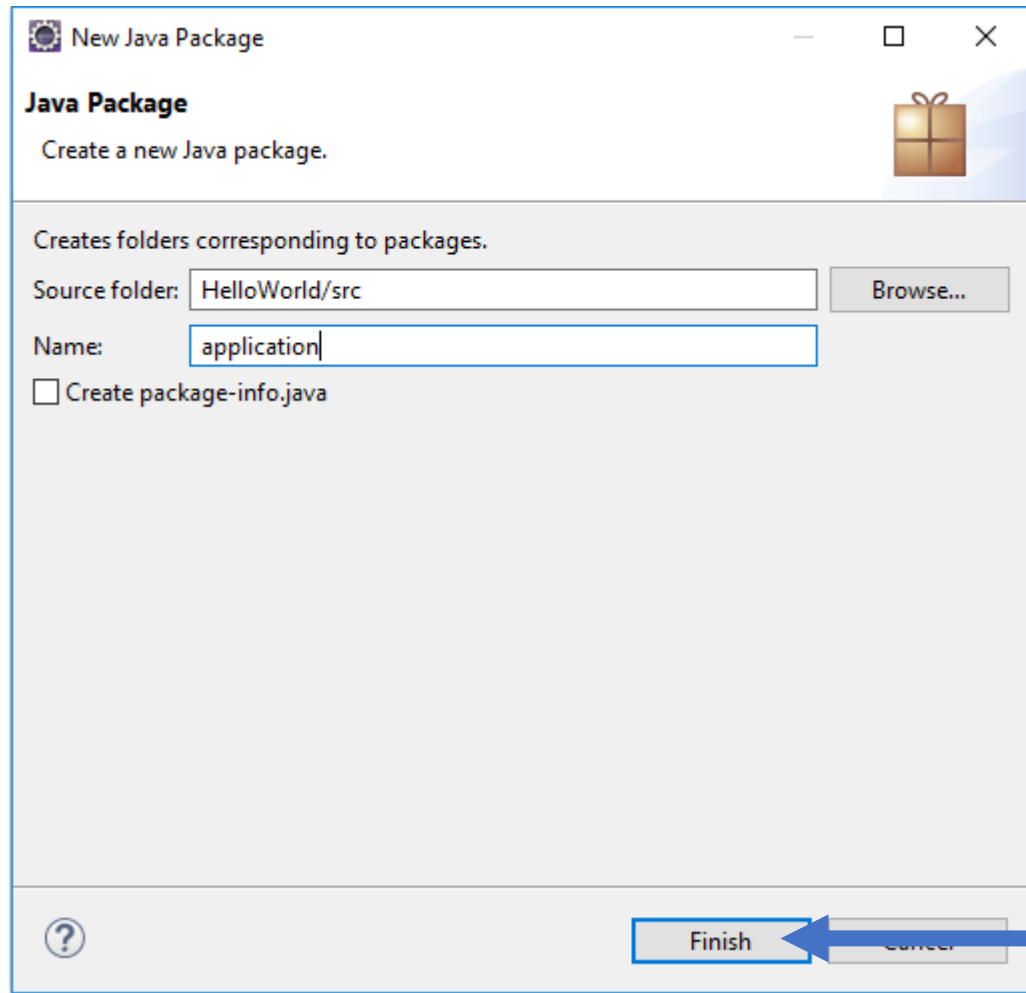
Click Finish button

This will show up under
Package Explorer tab



Select New then Left Click on Package



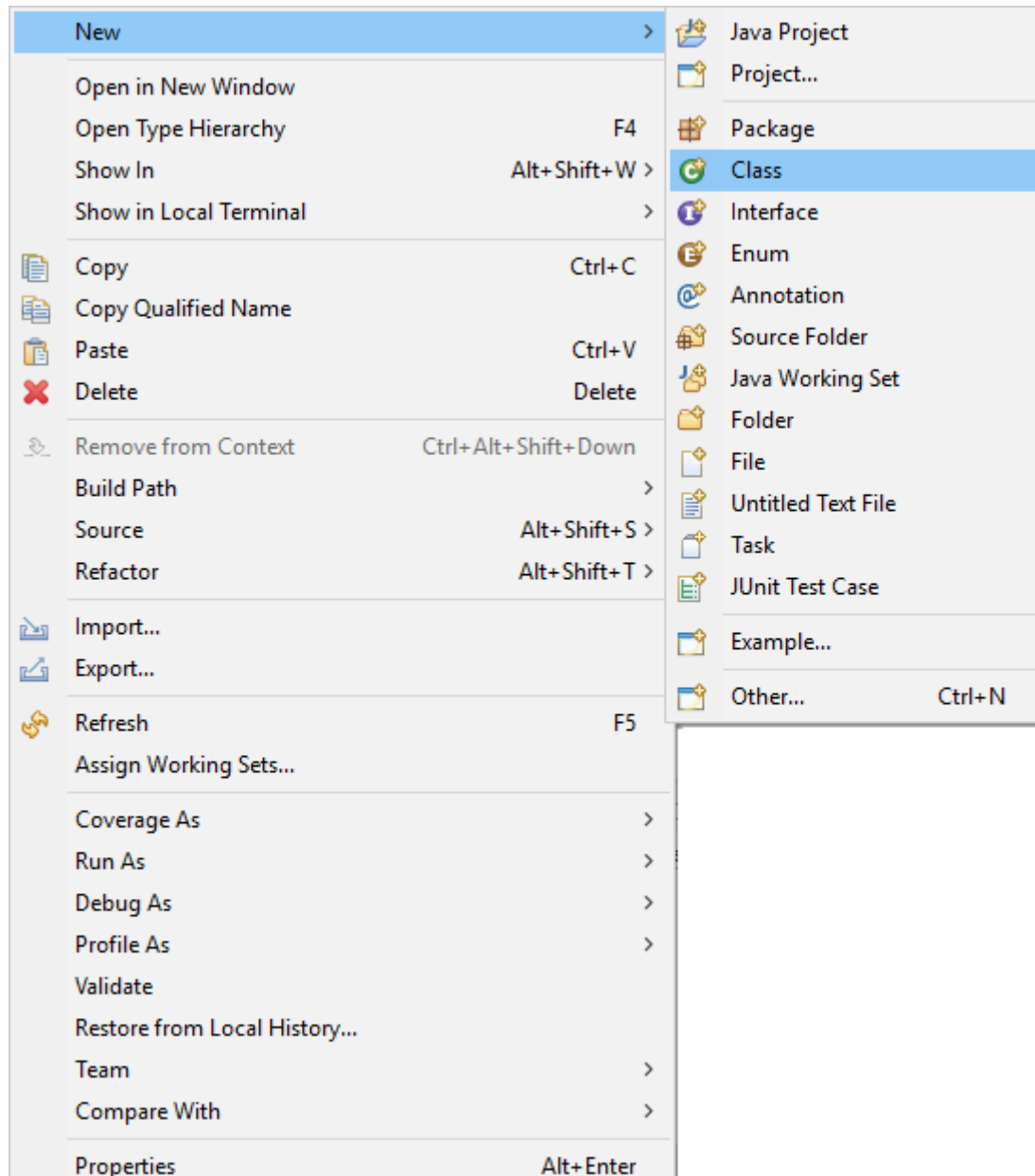


In Java We create packages in which we place our files containing our source code.

Convention to writing package names are lowercase letters only. Spaces and dashes(-) are not allowed. Instead use underlines (_) to separate words.

We will come back to packages and explain more later on in the course

After you specified the packagename click Finish



We need to write java code somewhere and we do that in "Classes".

Let's create a class inside of the package we created!

1. Right Click your package inside your project to bring up this menu.
2. Then left click Class

New Java Class

Java Class
Create a new Java class.

Source folder: HelloWorld/src Browse...

Package: application Browse...

☐ Enclosing type: Browse...

Name: HelloWorld

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Finish Cancel

Class names is by convention
named with UpperCamelCase style

1. Name your class in the
name field
2. Click Finish

Coding Hello World

Write the following code in your new class:

```
package application;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World");
    }

}
```

This program will print the text Hello World to the console

Hello World line by line

package application;

Package declaration is ALWAYS on line 1. Notice the semicolon (;) at the end.

public class HelloWorld {

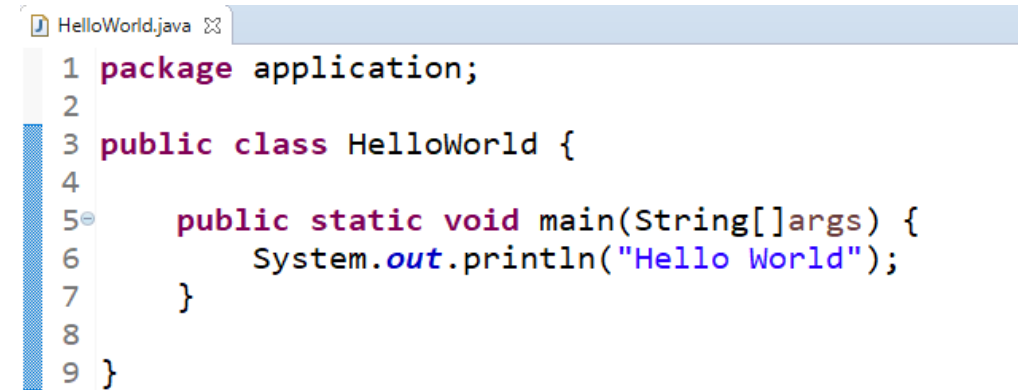
}

The class itself spans from line 3 to line 9. Notice the curlybracers ({ }) defining the *scope* of the class.

public static void main(String[]args) {

}

This is a special method called main. The program starts from the main method. And when at the end of the main method *scope* { } , the program ends.



```
1 package application;  
2  
3 public class HelloWorld {  
4  
5     public static void main(String[]args) {  
6         System.out.println("Hello World");  
7     }  
8  
9 }
```

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

This is a *statement* inside the main method.
Each individual *statement* HAVE to end with a semicolon (;)

Variables

Primitive data types

- Java has eight primitive datatypes.
- They are used for storing data in memory.
- What type of data they can store depends on the datatype.

These are the datatypes in Java:

Byte, short, int, long, float, double, boolean, char

Data types byte, short, int and long

- Byte short and int store whole numbers (integers) in different sizes.
- Int is the default whole number in java.

Data type	Min value	Max value	Bits	Default value (field)
byte	-128	127	8-bit	0
short	-32768	32767	16-bit	0
int	-2147483648	2147483647	32-bit	0
long	-9223372036854775808	9223372036854775807	64-bit	0

Data types float and double

- Both float and double are decimal numbers.
- Decimals in Java are written with a dot (.) instead of comma (,)
- Float is 32bit and double 64bit.
- Double is the default choice for decimals.

Data type boolean

- Only has two values: true or false
- Used for tracking if some condition has status true or false.

Data type char

- Char is a 16-bit Unicode character.
- Min value of '\u0000' or int representation of 0.
- Max value of '\uffff' or int representation of 65535

Creating primitive variables

- Its easy to create a primitive variable.

Data type Name



`int speed;`

This variable is unassigned.
`System.out.println(speed);`
would cause a compile error.

`speed = 100;`

Assigning the value of 100 to speed

`System.out.println(speed);`

Will work and print 100

To be able to access the value from a variable it must be assigned, but you dont NEED to give it a value at declaration if you dont want.

Some declaration examples

```
//Creates a byte with name b and value 10
byte b = 10;
//Creates a short with name s and value 10
short s = 10;
//Creates a int with name i and value 10
int i = 10;
//Creates a long with name l and value 10
long l = 10;
//Creates a float with name f and value 23.5
float f = 23.5f;
//Creates a double with name d and value 23.5
double d = 23.5;
//Creates a char with name c and value 'a'
char c = 'a';
//Creates a boolean with name bool and value true
boolean bool = true;
```

More declaration examples

```
//You can declare a variable without giving it a value  
double decimal;
```

```
//Create three unassigned variables with name a, b and c  
int a,b,c;
```

```
//What does this do?  
int a=1,b,c;
```

Operators

Assignment (=)Operator

- Very common operator that assigns the value of right hand [operand](#) to the left.

```
int number1 = 10;
```

```
int number2 = 5;
```

```
//number1 will be assigned the value of number2
```

```
number1 = number2;
```

Arithmetic Operators(+ - * /)

Addition:

```
int result = 42 + 10;
```

Subtraction:

```
int number1 = 5;  
int number2 = 10;  
int result = number1 - number2;
```

Multiplication:

```
int number1 = 10;  
int result = number1 * 34;
```

Division:

```
double result = 100 / 2.5;
```

“Changing” a primitive

```
//Trying to modify a number this way will not compile  
int number = 5;  
number + 1;
```

```
//In order to change a primitive you need to do this  
int number = 5;  
number = number + 1;
```

```
double decimal = 432.43;  
decimal = decimal - (234.32 / 32);
```

Increment (++) and decrement (--) operators

```
//Number increment by using ++  
int number = 0;  
number++;
```

==

```
//Same as using number++  
int number = 0;  
number = number + 1;
```

```
//Number decrement by using --  
int number = 0;  
number--;
```

==

```
//Same as using number--  
int number = 0;  
number = number -1;
```

```
int number = 0;  
//What will this print?  
System.out.println(number++);
```

?

```
int number = 0;  
//What will this print?  
System.out.println(++number);
```

Modulus (%) or the "Remainder operator"

```
//Since 9 is evenly divided by 3 modulus will be 0
```

```
int modulus = 9 % 3;
```

```
/*
```

```
 * Since 11 is not evenly divided by 3 we have a modulus value of 2
```

```
 * Since modulus 9%3 is 0, the "remainder" is 2. (9 + 2 = 11)
```

```
*/
```

```
int modulus = 11 % 3;
```

```
//What will modulus be here?
```

```
int modulus = 10 % 3;
```

```
//..or here?
```

```
int modulus = 12 % 3;
```


Comparison Operators: (==, !=, <, >, <=, >=)

Needed to evaluate if some condition is true or false.

Equality operator == is used to check if two primitives are equal

```
int number1 = 1+2+3+4;           //10
int number2 = 3+2+1+4;           //10
boolean isEqual = number1 == number2; //true
System.out.println(isEqual);      //prints true
```

Next we are using != which will be true if two primitives are not equal


```
char letter1 = 'A';
char letter2 = 'a';
boolean isNotEqual = letter1 != letter2; //true
System.out.println(isNotEqual);          //print out true
```

Comparison Operators: (==, !=, <, >, <=, >=)

```
int number1 = 10, number2 = 11;  
System.out.println(number1 < number2);           //true  
System.out.println(number1 > number2);           //false  
  
System.out.println(10 < 10);                      //false  
System.out.println(10 <= 10);                     //true  
  
System.out.println(12 % 3 > 0);                   //false
```

Selections

if keyword Paratheses (required)



```
if(booleanExpression) {  
    //Branch if true  
}
```

If statements

- Often we want our code to branch.

Given the following example. If a person is older than 18. He/She could have a beer.

```
int age = 20;  
if(age >= 18) {  
    System.out.println("You can have a beer");  
}
```

If – else statements

- What if we wanted to print out something if a person is below 18?
- The solution for that is using else.

```
int age = 17;

if(age >= 18) {
    System.out.println("You can have a beer");
} else {
    System.out.println("You are too young to drink beer");
}
```

If – else if statements

- To have more than two branches you can use this technique.

```
int age = 17;

if(age >= 18) {
    System.out.println("You can have a beer");
}else if(age < 18 && age >= 15){
    System.out.println("You can drive a moped but not drink beer");
} else {
    System.out.println("you are too young to drive a moped and drink beer");
}
```

Notice the two ampersands (&&). Both boolean expressions needs to be true for the code to execute

Selection switch statement

- Decision making structure
- Single value is evaluated and redirected to first matching *case*.
- If no matching case is found an optional *default case* is called.
- Supported evaluation types are: *byte, short, char, int, String and enum values*.

Switch Example

```
double num1 = 5, num2 = 7, result = 0;  
char operator = '+';
```

char operator is evaluated

```
switch(operator) {
```

```
case '+':
```

```
    result = num1 + num2;
```

```
    break;
```

```
case '-':
```

```
    result = num1 - num2;
```

```
    break;
```

```
case '*':
```

```
    result = num1 * num2;
```

```
    break;
```

```
case '/':
```

```
    result = num1 / num2;
```

```
    break;
```

```
default:
```

```
    System.out.println(operator + " is not supported");
```

```
}
```

```
System.out.println(result);
```

cases are +, -, * and /

Break statements in all cases to get out of the switch

Default statement will trigger if no matching case is found

Java Strings

Java Strings

- Often referred to as the ninth primitive.
- It's NOT a primitive. All Strings are objects.
- Often treated as primitives
- It's a sequence of characters.
- Notice the capital 'S' in String.

```
String myName = "Erik Svensson";
```

```
System.out.println(myName);
```

Immutable. (Not able to change)

- Strings are immutable. Which means they **don't** change (mutate)

```
//Declaring the String and assigning value
String name = "Erik";
//Adding " Svensson" to the name String by calling the String method concat
name.concat(" Svensson");
//What will this print?
System.out.println(name);
```

Whenever you try to modify a String it gives you a new String

Immutable

```
String name = "Erik";  
name = name.concat(" Svensson");  
System.out.println(name);
```

```
String name = "Erik";  
name = name + " Svensson";  
System.out.println(name);
```

"Here the concatenation will work. Do you notice why?"

Building Strings

```
String firstName = "Erik";  
String lastName = " Svensson";  
int age = 42;  
String hobby = "Programming";
```

```
String sentence = firstName + lastName + " is " + age + " years old and likes " + hobby;
```

```
System.out.println(sentence);
```

Having basic knowledge about Strings we can build them like above example

"There is a lot to learn about the String class. We will explain it more in later lectures"

Objects

Introduction to classes and objects

What if i had many persons that have names and age and hobby, like in last example?

Notice the variable names. We can't have duplicate variable names in the same scope.

This solution will get complicated really fast

```
String firstName = "Erik";  
String lastName = " Svensson";  
int age = 42;  
String hobby = "Programming";
```

```
String firstName2 = "Anna";  
String lastName2 = " Andersson";  
int age2 = 54;  
String hobby2 = "sport cars";
```

```
String firstName3 = "Nils";  
String lastName3 = "Björk";  
int age3 = 27;  
String hobby3 = "gaming";
```

Introduction to classes and objects

We need a reusable blueprint to store our data in.

We need a single representation that groups our variables and String objects

We need each Person to be unique and carry its own variables and functionality with it

What we REALLY need is to create a CLASS that serves as a blueprint for OBJECTS of that class.

```
String firstName = "Erik";  
String lastName = " Svensson";  
int age = 42;  
String hobby = "Programming";
```

```
String firstName2 = "Anna";  
String lastName2 = " Andersson";  
int age2 = 54;  
String hobby2 = "sport cars";
```

```
String firstName3 = "Nils";  
String lastName3 = "Björk";  
int age3 = 27;  
String hobby3 = "gaming";
```


Creating a class and Instantiation

```
class Person {  
  
    String firstName;  
    String lastName;  
    int age;  
    String hobby;  
  
}
```

```
//Create object of Person like this  
//Also known as INSTANTIATE an object  
Person erik = new Person();
```

Type Object name Assignment operator Call to constructor

Assigning values to the Class fields

Next we assign values to our object

```
class Person {  
    String firstName;  
    String lastName;  
    int age;  
    String hobby;  
}  
  
Person erik = new Person();  
erik.firstName = "Erik";  
erik.lastName = "Svensson";  
erik.age = 42;  
erik.hobby = "programming";
```

A diagram consisting of four horizontal blue arrows pointing from the class field declarations on the left to the corresponding object field assignments on the right. The arrows connect 'firstName' to 'erik.firstName', 'lastName' to 'erik.lastName', 'age' to 'erik.age', and 'hobby' to 'erik.hobby'.

Implementing functionality

Next we create a method that return a String describing our object

```
class Person {  
  
    String firstName;  
    String lastName;  
    int age;  
    String hobby;  
  
    public String getInformation() {  
        return firstName + " " + lastName + " is " + age + " years old and likes " + hobby;  
    }  
}
```

Summary

- Learned how to create a Java Project in Eclipse.
- How to print out variables and Strings in the console.
- Introduction to primitive variables in Java.
- How to use a variety of operators in Java.
- Make Selections with If and Switch statements.
- Learned the basics about Java Strings
- Learned how to create a simple Class
- Object instantiation from a Class

Practices:

- Create a String with your first name as content. Print out the String object to the console.
- Create two Strings. One with your first name and the other with your last name. Print out both Strings to the console with a " " separating them.
- Create two variables with decimal values. Create a third decimal variable called result. Assign the sum of the two first variables to result.

Practices:

- Create a class called Car. The class should contain the following information. Car brand, registration number, max speed and owner name. Instantiate an Object of the class and assign values to the object.
- Make an integer called score and assign a value to it. Create an if – else statement that should print out "you passed" if the score is above or equals to 65. If score is below 65 it should print "you did not pass".

Practices:

- Being a developer you need to learn how to search for information on the Internet. Open up a web browser and search for "java input from user".
 - Use your new knowledge to take a String as input from the user and store it in a String object.