Our first application

Reading input
String formating
Parsing input
Escaping
Code comments

Hello World in Java (1)

The code below is your typical starting point when creating a new Java project:

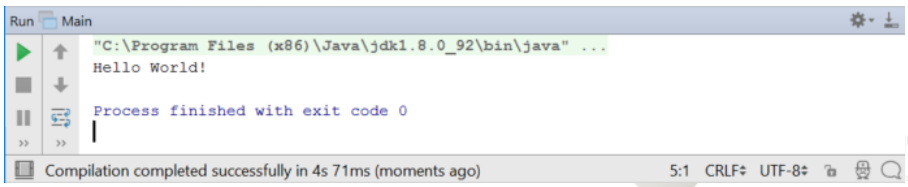
```
public class Main {
    public static void main(String[] args) {
        // write your code here
    }
}
```

Hello World in Java (2)

We'll start by simply sending "Hello World" to the console:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Run your program by pressing **Shift+Alt+F10**, or by selecting **Run → Run**



Reading input (1)

Let's do something more interesting

Can we get the application to accept our name as input, and write that instead of "world"?

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Something like:

```
Enter your name: Java
Hello Java!
```

Reading input (2)

To read input from the user we have to use the **Scanner** class and call the **NextLine()** method.

We need to import the **Scanner** class.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in  = new Scanner(System.in);
        in.nextLine();
        System.out.print("Hello World!");
    }
}
```

What else do we have to do?

Reading input (3)

This is a perfect place to utilize **IntelliSense**, to see what's available to us:

```
package com.company;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in  = new Scanner(System.in);
        System.out.print("Hello World!");
        String name = in.
    }
}
```

findInLine (Pattern pattern)	String
findInLine (String pattern)	String
findWithinHorizon (Pattern pattern, int horizon)	String
findWithinHorizon (String pattern, int horizon)	String
next ()	String
next (Pattern pattern)	String
next (String pattern)	String
nextLine ()	String
toString ()	String
close ()	void

Press Ctrl+Period to choose the selected (or first) suggestion and insert a dot afterwards >>

Tip: if you're not getting IntelliSense automatically, try invoking it manually by pressing **Ctrl+Space**.

Indicates that a String is **returned** to the caller.

Reading input (4)

A string is returned from the **NextLine()** method.

Let's create a **String** variable and store the result:

Notice the uppercase S in String

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in  = new Scanner(System.in);
        System.out.print("Hello World!");
        String name = in.nextLine();
    }
}
```

Reading input (5)

Let's print out the "Enter your name" prompt:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in  = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        System.out.println("Hello World!");
    }
}
```

System.out.print does not add a line break after writing, making our output a little more aesthetically pleasing.

Writing the input to the screen

Let's add the user-input to our message:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in  = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        System.out.println("Hello " + name + "!");
    }
}
```

Notice that we're using the **addition operator** in order to concatenate (join) strings.

String formatting

String formatting

Let's look at the last line of code in more detail:

```
System.out.println("Hello " + name + "!");
```

When concatenating strings and variables in this manner, code can quickly become cluttered

Using string formatting instead, with the **printf** command:

```
System.out.printf("Hello %s!\n", name);
```

Notice: **printf** does not add a newline! Put **\n** in the string for a newline.

String formatting

More possible string format specifiers:

Format specifier	Description
%d	Decimal integer
%f	Float, real number
%s	String
%c	Character
%t	Date/Time

Many more of these exist and are described in Oracle's Java Docs.
https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html

String formatting

We can perform additional formatting with these specifiers by using parameters.

Formatting Type	Which specifiers?	Input	Output
%10s	All	"hello"	`     hello`
%-10s	All	"hello"	`hello     `
%010d	%d and %f	3295	`0000003295`
%.2f	%f	56.497	`56.50`
%10.2f	%f	56.497	`     56.50`

These can be combined, e.g. %010.2f will be 10 characters long with 2 decimal places, with trailing zeroes if required.

Parsing input

Parsing input

If we want a number instead, we can prompt the user for more input the same way.

```
public class Main {
    public static void main(String[] args) {
        Scanner in  = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();

        System.out.printf("Hello %s!", name);
        System.out.println();

        System.out.print("What year were you born?: ");
        String year = in.nextLine();
    }
}
```

Parsing input

If we want to do math, we need to convert it to a number.

```
Scanner in = new Scanner(System.in);
String value = in.nextLine();

int i = Integer.parseInt(value);      // Returns value to an integer
```

ParseInt will take a string and convert it to an integer, like:

```
int x = Integer.parseInt("12345");
```

Parsing input

To calculate the age of the user we can write:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in  = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();

        System.out.printf("Hello %s!", name);
        System.out.println();

        System.out.print("What year were you born?: ");

        String birthYearInput = in.nextLine();
        int year = Integer.parseInt(birthYearInput);
        int age = 2013 - year;
    }
}
```

It's not 2013!
How can we solve this?

Use Libraries

We need to ask the system for the current year to fix the date issue.

In this case, we'll use the **LocalDate** class. This lets us ask for the **current date** when the application is run.

Just like with **Scanner**, we **import** the **LocalDate** class and update the age calculation formula:

```
import java.time.LocalDate;

int age = LocalDate.now().getYear() - year;
```

Now our code will always use the correct date!

Use Libraries

LocalDate is a new addition to Java 8.

If you're using Java 7 or earlier, use the **Calendar** class.

```
import java.util.Calendar;

int age2 = Calendar.getInstance().get(Calendar.YEAR) - year;
```

As the Java language have evolved during its lifetime, you will find many ways to do the same thing.

Final Code

Finally, we print the name and age to the screen:

```
import java.time.LocalDate;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in  = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();

        System.out.print("What year were you born?: ");
        String birthYearInput = in.nextLine();
        int year = Integer.parseInt(birthYearInput);
        int age = LocalDate.now().getYear() - year;

        System.out.printf("Hello %s! You are %d years old!",
                          name, age);
        System.out.println();
    }
}
```

Notice that we can split long statements over several rows.

Refactoring the Code

We only use the variable **BirthYearInput** in one place, so let's refactor and remove it!

```
System.out.print("What year were you born?: ");
String birthYearInput = in.nextLine();
int year = Integer.parseInt(birthYearInput);
```

```
System.out.print("What year were you born?: ");

int year = Integer.parseInt(in.nextLine());
```

in.nextLine() is all we care about.

So we'll just move it here!

Refactoring the Code

Scanner can also convert to int using nextInt()

```
System.out.print("What year were you born?: ");

int year = Integer.parseInt(in.nextLine());
```

```
System.out.print("What year were you born?: ");

int year = in.nextInt();
```

How are these different

Escaping

Escaping characters

Some characters are not straightforward to print, such as the following:

Quotation marks are used to delimit strings. How do we print them inside a string?

```
Did someone say "Hello World?"
```

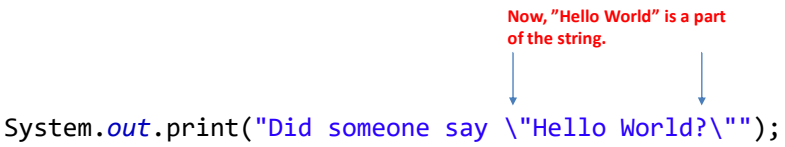
If we try to use this in Java, the code will not compile. How can we get it to print?

To Java, Hello World is not part of the string.

```
System.out.print("Did someone say "Hello World?"");
```

Escaping characters

Simple: We use a **backslash** to escape the character:



Several other special characters can be written this way as well:

Character	Description
\\	Write ”\”
\t	Tab
\n	Newline

There are more of these, and you can find them here:
http://docs.oracle.com/javase/tutorial/java/data/characters.html

Comments

Comments

Before we get started on writing code, you should know how to comment your source code.

1
```
// This comment spans one line
// This is a second line also starting
// with double slashes
```

2
```
/* This comment
   spans several
   lines */
```

3
```
int a = 2; // This is a valid comment
```

4
```
int a = 2; /* As is this! */
```

JavaDoc Comments

Type /** and press enter

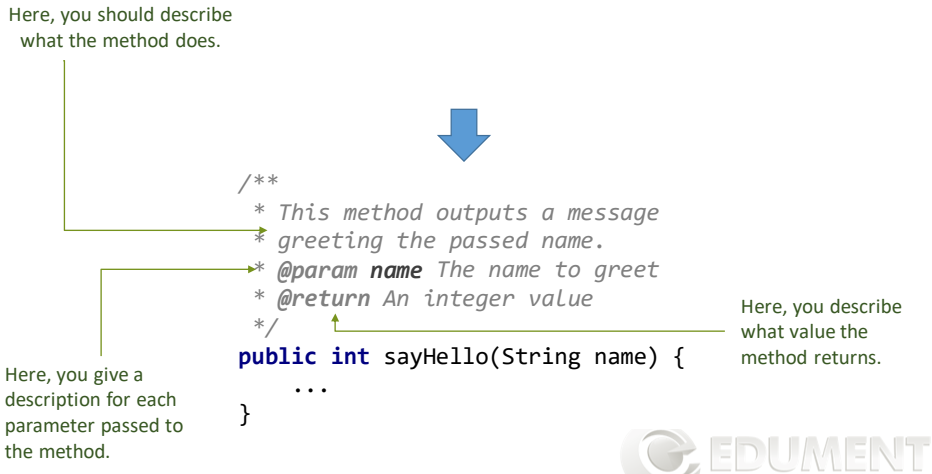
Auto generated method comments

```
/**
public int sayHello(String name) {
    ...
}
```

```
/**
 *
 * @param name
 * @return
 */
public int sayHello(String name) {
    ...
}
```

JavaDoc Comments

You should then fill in the JavaDoc comment with the appropriate information.



Exercise 3

Let's do exercise 3