AP Computer Science A

UNIT 1 TOPIC 1
Java syntax, Comments & println vs. print

Goals for Today

- Understand and analyze Java **syntax rules**, style conventions, and **compiler errors**
- Understand how to add **comments** to code
- Understand what the difference is between `print` and `println`

True or False?

WARM UP! With your partner!

- *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- Every **class** has its own **.java file**, with the same name as the class.
- The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- **`public`**, **`static`**, **`class`**, and **`void`** are all reserved Java **keywords**.
- **`System.out.println()`** is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- Capitalization does not matter in Java.

True or False?

- T *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- Every **class** has its own **.java file**, with the same name as the class.
- The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- `public`, `static`, `class`, and `void` are all reserved Java **keywords**.
- `System.out.println()` is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- Capitalization does not matter in Java.

True or False?

- T *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- T Every **class** has its own **.java file**, with the same name as the class.
- The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- **`public`**, **`static`**, **`class`**, and **`void`** are all reserved Java **keywords**.
- **`System.out.println()`** is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- Capitalization does not matter in Java.

True or False?

- T *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- T Every **class** has its own **.java file**, with the same name as the class.
- T The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- **`public`**, **`static`**, **`class`**, and **`void`** are all reserved Java **keywords**.
- **`System.out.println()`** is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- Capitalization does not matter in Java.

True or False?

- T *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- T Every **class** has its own **.java file**, with the same name as the class.
- T The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- T `public`, `static`, `class`, and `void` are all reserved Java **keywords**.
- `System.out.println()` is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- Capitalization does not matter in Java.

True or False?

- T *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- T Every **class** has its own **.java file**, with the same name as the class.
- T The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- T **`public`**, **`static`**, **`class`**, and **`void`** are all reserved Java **keywords**.
- T **`System.out.println()`** is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- Capitalization does not matter in Java.

True or False?

- T *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- T Every **class** has its own **.java file**, with the same name as the class.
- T The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- T `public`, `static`, `class`, and `void` are all reserved Java **keywords**.
- T `System.out.println()` is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- F Capitalization does not matter in Java.

True or False?

- T *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- T Every **class** has its own **.java file**, with the same name as the class.
- T The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- T `public`, `static`, `class`, and `void` are all reserved Java **keywords**.
- T `System.out.println()` is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- F Capitalization ***does*** ~~not~~ matter in Java.

True or False?

- T *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- T Every **class** has its own **.java file**, with the same name as the class.
- T The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- T `public`, `static`, `class`, and `void` are all reserved Java **keywords**.
- T `System.out.println()` is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- F Capitalization ***does*** ~~not~~ matter in Java (Println vs. println)

True or False?

- T *Every* Java program must have at least one **class**, a **main method**, and at least one **statement** (command).
- T Every **class** has its own **.java file**, with the same name as the class.
- T The **main method** is the starting point for *every* Java program and *always* looks like:

```
public static void main(String[] args)
```

- T `public`, `static`, `class`, and `void` are all reserved Java **keywords**.
- T `System.out.println()` is a Java method that prints whatever is in the ( ) to the console, then moves the cursor to the next line.
- F Capitalization ***does*** ~~not~~ matter in Java (Println vs. println, System vs. system)

Agenda

- Demo: Syntax Rules, how to add comments
- U1L1: print vs. println

Turn and Talk!

- Discuss each:
  - **Use of various punctuation marks:** Can you find six different punctuation marks in this program? How do you think each is used?
  - **Words that are capitalized vs. words that are not:** What "rules" do you think might be in place for capitalization?
  - **Use of indentation:** How and why do you think it's used?

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

DEMO: Code Style, Syntax & Compiler Errors

Bugs are a *natural* part of programming; you *will* make errors and you will make them often! You should get comfortable making and learning from mistakes -- it is the single BEST way to learn to program! 😎

With your partner: Let's bug bash!

This code contains multiple syntax/compiler errors and two "code style convention violations"

How many can you spot, just by scanning the code and using your eyes and brains only!

```
public class main {
public Static main(string[] args) {
system.out.println(Hello world again!)
}
```

Bug bash!

1. Open up Replit, log in, go to our class team, and open the **U1T1 Lab** project
2. Look at the code and **hover your mouse over each red and orange squiggly** to see information about each **syntax error.** These squigglies are produced by Replit as it *attempts* to compile your code, but it can't due to **syntax errors!** This is why **syntax errors are considered "compiler errors"** since they prevent your code from being compiled (and if you try to *run* the program, it won't work):

```
1 ▼ public class main {
2 ▼ public Static main(string[] args) {
3   system.out.println(Hello world again!)
4   }
```

1. Did you correctly identify all syntax errors?!
2. **Fix all** syntax/compiler errors **as well as** the any code style violations.
3. When you think you have them all fixed, **execute/run** your program; if you got them all, you should see:

```
Hello world again!
```

Solution

```
public class main {
public Static main(string[] args) {
system.out.println(Hello world again!)
}
```

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

Replit uses red squigglies to show you compiler/syntax errors that need fixing *before* your code will run:

syntax/compiler error fixed
code style violation fixed

DEMO: Comments in Java

Quick Activity: Try it out!

Add three different comments in your

1. Try adding a single-line comment with `//`
2. Try adding a multi-line comment with `/*  */`
3. Try using `//` to “comment out” a line of code so it doesn’t execute when you run the code (this is great for debugging!)

Lab Launch

Below are two **code segments**. A **code segment** is a snippet of code that is used for discussion/analysis, and you should assume any code snippet was taken from a working program (just omitting the parts of the program that aren't important for the discussion, such as the class or main method)

Displaying info with **`println`**

```
System.out.println("Look");
System.out.println("at me!");
System.out.println("Hi!");
```

Displaying info with **`print`**

```
System.out.print("Look");
System.out.print("at me!");
System.out.print("Hi!");
```

PREDICT! Do you think the output from these two code snippets will be different? If so, how?

We aren't testing these now! In today's Lab, you will get to test your prediction, and get to know the two commands, **`System.out.println`** and **`System.out.print`** very well!

Syntax Rules, Conventions & Compiler Errors

Basic Java Code Style & Syntax

- **Syntax Rules**, i.e. "the grammar rules of Java" (**required, enforced by compiler**):
  - **Curly brackets (or braces)** `{ }` are used to enclose the code of all **classes** and **methods**
    - *These indicate where a certain block of code starts and stops!*
  - **Square brackets** `[ ]` are used for arrays
  - **Parentheses** `( )` are used for providing arguments to methods (as well as mathematical expressions)
  - All Java **statements** (a.k.a. **commands**), like line 5, end in a **semicolon** `;`
  - **String literals (text typed between two quotes)** are enclosed inside **double quotes** `" "`
    - **"Hello world!"** is an example of a **string literal**
  - **Periods ("dots")** are used to call methods from *other* classes
    - **System** is a class that is part of the Java library; the "dot" is how we use the *println* method

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

Basic Java Code Style Conventions

- Code Style Conventions (*not* enforced by compiler, but it's bad style to violate!):
  - All class names get capitalized by convention (to distinguish them from methods, which are lowercase)
    - "HelloWorld," "System," and "String" are all class names
    - There are only three times capitalization is used in Java (classes and two others, which we will discuss later); everything else, like methods and keywords, are lowercase!
  - Use indentations to help show how code is structured ("levels" of code)
    - IDEs often auto-indent for you

Proper indentation & class capitalization ☺

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

No indentation, *technically* OK, but hard to read code! Don't do this! ☹

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

You might be wondering: if System and String are both *classes*, then where are the System.java and String.java files? The answer is that these are "core classes" that are part of Java, rather than a class that *we* are writing, so we don't have access to their .java files (but they do exist!)

Basic Java Code Style & Syntax

- "Up to you" code style (*either is acceptable -- it's a personal preference)*
  - Opening brace on a new line or not?
  - Indentation size
    - **Tab** size of **2 - 4 spaces** is common; no value is "standard" (Replit is set to have **Tab** equal 4 spaces by default)

Opening braces on new lines (you will see this style used on some AP questions)

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

The way we will do it (since it's how the pros do it)

Opening braces *not* on new lines (more common in the professional software industry)

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

Syntax/Compiler Errors (Bugs!)

Replit uses red squigglies to show you compiler/syntax errors that need fixing *before* your code will run: 

Red squigglies under code indicates **compiler error**, also called a **syntax error**.

- Replit **auto-compiles** as you type, immediately checking your code for proper **syntax**; if you still try to **run/execute** your code while there are unfixed syntax errors, Replit gives error messages.
- Syntax errors are examples of "**compile-time errors**" because they are found when the code is being compiled, which Replit does automatically as you type (we will see examples of "*runtime errors*" later).
- Replit tries to give you helpful information about the error:

Which .java file and line number

Details about the problem

```
Main.java:5: error: ';' expected
        System.out.println("Hello, World")
                                           ^
```

- Syntax/compiler errors prevent the code from being successfully compiled *-- and you won't be able to successfully run the program without fixing **all** errors!*
- Any type of programming error is called a **bug**; finding & fixing errors is called **debugging**.

Comments in Java

Comments in Java

- **Comments** are used to explain code and to make it more readable, and are also often used to prevent code from executing when testing (i.e. by "commenting out" certain code you don't want to run).
- **Comments** are *completely ignored by the compiler* (in other words, comments are never compiled or executed).
- There are **three ways** to add comments in Java:
  - Using `/*` and `*/` for **single-** or **multi-line comments**
  - Using `//` for **single-line comments**
  - Using `/**` and `*/` for automating **Javadoc documentation** (*we will see and discuss this in Unit 5*)

} We will use these two for now

```
/* Here is a
   multi-line comment
   using asterisks! */

public class Main
{
  /* here is a single line code with asterisks */
  public static void main(String[] args)
  {
    // Here is a single line comment!
    System.out.println("Hello world!");
    System.out.println("Look at these comments!"); // Here is another!
    //System.out.println("Don't print me!");
  }
}
```

Preventing a line of code from running

More Comment Examples

Which style of comment style to use is up to you!
You will see them both ways throughout this course.

Multi-line comment with /* and */ (formatted to look nice 😊)

Single-line comment with /* and */

Multi-line comment with /* and */

Using a // comment to "disable" certain code for testing

```
/* Programmer: Mr. Miller
 * Date: 9-3-20
 */

public class Main
{
   /* this method begins the entire program */
   public static void main(String[] args)
   {
      /* The code segment below prints
      out "What a crazy World! followed
      by "Hello!" on a new line */
      System.out.print("What ");
      System.out.print("a crazy ");
      System.out.println("world!"); // prints "world!" and a new line
      System.out.println("Hello!");
      // System.out.println("test123");
   }
}
```

Single-line comment with //

Java `print` & `println` commands and String Literals

print vs. println

- `System.out.print()`

  Display whatever is in the ( ) on the screen, then **wait** at the end of the line

- `System.out.println()`

  Display whatever is in the ( ) on the screen, then **move the cursor to the next line** (the "`ln`" in `println` stands for, you guessed it, line!)

  We sometimes call this command the "print line" command.

print vs. println: line-by-line example

Code segment:

```
System.out.print("To be ");
System.out.print("or not to be.");
System.out.print("That is ");
System.out.println("the question.");
System.out.print("This is");
System.out.println(" for the whole family!");
```

Direction of code execution

Console output & location of cursor *after* the line executes:

After 1st line executes:
```
To be
     ^
```

After 2nd line executes:
```
To be or not to be.
                   ^
```

After 3rd line executes:
```
To be or not to be.That is
                          ^
```

After 4th line executes:
```
To be or not to be.That is the question.

^
```

After 5th line executes:
```
To be or not to be.That is the question.
This is
       ^
```

After 6th line executes:
```
To be or not to be.That is the question.
This is for the whole family!

^
```

`println` displays information and ***then*** *goes to the next line!*
`print` displays information and then *waits*.

String Literals

- Everything we have typed so far inside the ( ) for both **`print`** and **`println`** have been **string literals**:
- A **string literal** is any sequence of characters (letters, numbers, and/or symbols) *typed in between two quotes*.

```
System.out.print("Look ");
System.out.println("at me!");
System.out.println("Hi!");
```

These **are** all string literals because they are all sequences of characters typed *between two quotes!*

```
System.out.print(Look );
System.out.println(at me!);
System.out.println(Hi!);
```

These are **not** string literals because none of these are between quotes! Trying to print these out will cause compile errors!

String Literals

More Examples:

- `"AP CSA"`
- `"This is a string literal!"`
- `"A4687BC-blahblah-1!2$%$"`
- `"* * * BOOM! * * *"`
- `"1 + 2"`
- `"System.out.println"`
- `"baad speling"`

These ARE all string literals because they are all sequences of characters typed *between two quotes!*

More NON-Examples:

- `AP CSA`
- `This is NOT a string literal!`
- `A4687BC-blahblah-1!2$%$`
- `* * * BOOM! * * *`
- `1 + 2`
- `System.out.println`
- `baad speling`

These are NOT string literals because a string literal in Java is any sequence of characters *between two quotes*, and none of these are between quotes!

Summary

- There are **syntax rules** which are required and "enforced" by the compiler, including using `{ }` around classes/methods and using semicolons at the end of every Java statement.
- Replit's **compiler** auto-checks for **syntax/compiled errors** as you type your code; if any are found, it puts red squigglies under the problematic syntax. If you try to run your program when there are compiler/syntax errors, Replit produces error messages.
- Programming errors (**bugs**) and fixing them (**debugging**) are both important parts of the learning process!
- **Comments** are used to explain code and to make it more readable and are completely ignored by the compiler.
- There are *three* ways to add **comments** (shown on slides 29-30).
- `System.out.print()` displays whatever is in the ( ) on the screen, then **waits** at the end of the line. `System.out.println()` (often pronounced "print line") displays whatever is in the ( ) on the screen, *then* **moves the cursor to the next line.**
- A **string literal** is **any** sequence of characters (letters, numbers, and/or symbols) **typed in between two quotes.**