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

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

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

```java
// Class not intended to be used to create other classes

public final class Main {

   // All parameters should be final; enforced by 'checkstyle
   public static void main (final String[] args) {
      System.out.println ("Hello world!");
   }
}
```

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

Consider each word/token.

- What happens if you leave it out?
- What happens if you modify it?

- access modifier for things with unrestricted access; one public, top-level Java class per file
- modifier for classes that are not to be sub-classed
- keyword introducing a Java class
- name of class; capitalized by convention; should be same as file name
- access modifier for methods with unrestricted access; method `main` must be public if it is to be the starting point of the program by the Java virtual machine
- method modifier indication a non-instance method
- return type of void means the method does not return a value; it is a subprocedure not a function

- name of method; must be called `"main"` if it is to be the starting method (entry point)
- type of the one parameter to the method; must be an array of strings (or equivalently varargs), if the method is to be the starting method
- name of the one parameter to the method
- `java.lang.System` is the name of the class in package `java.lang` containing standard I/O objects
- Field of `java.lang.System` with type `java.io.PrintWriter` containing the object with the reference the program's standard output stream.
- Name of the overloaded method that puts strings on to output stream (prints or displays the text on the screen).

Definitions.

- access modifier (e.g., public, private)
- entry point (starting point of execution)

Rules of thumb.

- Declare your (outer) classes public.
- One public class the same name as the file.
- Declare your classes final.
- Declare your formal arguments final.
- (No public constructors for utility classes.)

In Java 21, the entry point need not be static, need not be public, and need not have a `String[]` parameter. Then we can simplify the “Hello, World!” program to:

```java
class HelloWorld {
    void main() {
        System.out.println("Hello, World!");
    }
}
```

Java 21 introduces unnamed classes to make the class declaration implicit:

```
void main() {
    System.out.println("Hello, World!");
}
```

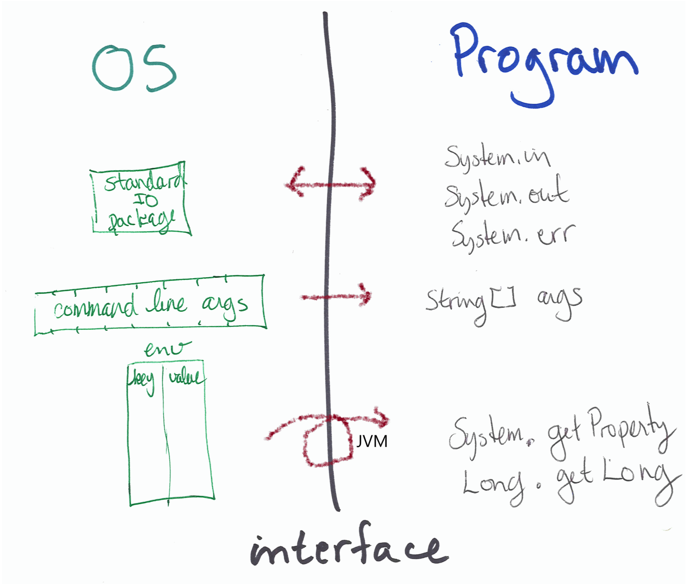
```
String greeting = "Hello, World!";

void main() {
    System.out.println(greeting);
}
```

A source file named HelloWorld.java containing an unnamed class can be launched with the source-code launcher, like so:

```
$ java HelloWorld.java
```

The Java compiler will compile that file to the launchable class file HelloWorld.class. In this case the compiler chooses HelloWorld for the class name as an implementation detail, but that name still cannot be used directly in Java source code.


OS
Program
standard IO package
System.in
System.out
System.err
command line args
String[] args
env
key
value
JVM
System.getProperty
Long.getLong
interface

OS Interface

- standard IO package; abstract and standardized
- command line arguments; simple
- environment map and also JVM properties; many OS dependent keys

- `System.in`, `System.out`, `System.err`
- `String[] args`
- `System.getEnv()`, `System.getProperties()`

JVM has it own platform environment established in negotiation with the OS.

```
// The (unmodifiable) key,value pairs  of OS environment
Map<String, String> env = System.getenv();

// Less platform dependent are the Java system properties
System.getProperties().list(System.out);
```

```
// A program can get values from Java system properties
System.out.println (System.getProperty ("file.encoding"));

// A program can set Java system properties
// Be careful arguing with the OS
System.setProperty ("file.encoding", "Latin-1");

// System properties sometimes 'travel' invisibly to
// where they are needed, at indeterminate times.
System.out.println (Charset.defaultCharset());
```

Properties can be set on the command line. But they may not be set, so providing defaults is a good idea.

```
// User can supply environmental values (Unix example):
// * with JVM args             '$ java -Dkey=12345 Main'
// * with OS environment, eg, '$ env key=12345 java main'
String seed1 = System.getenv().getOrDefault("key","54321"));
String seed2 = System.getProperty ("key", "54321");
```

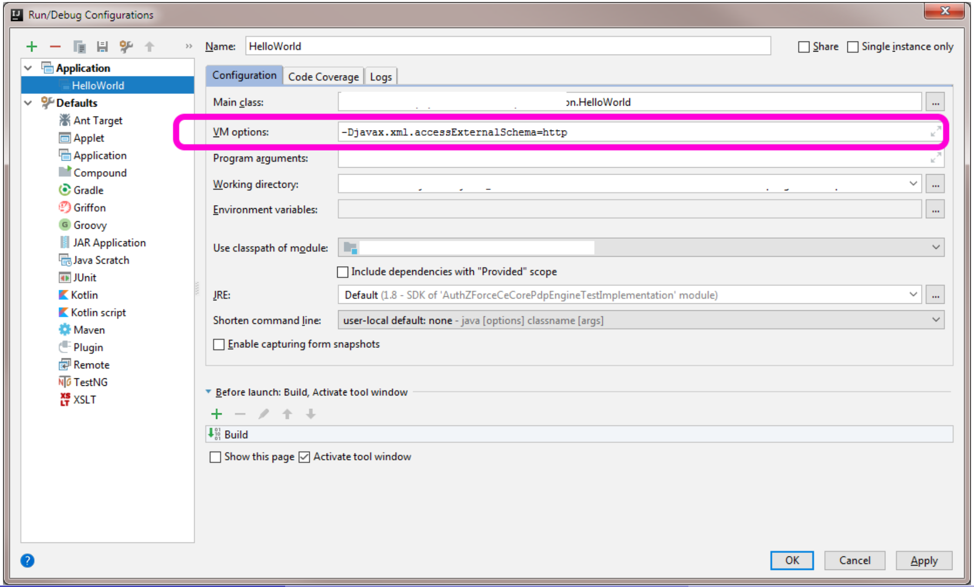
Use Case: Random Numbers

```
// User can supply environmenal values:
// * with JVM args '$ java -Dseed=12345 Main'
// * with OS environment, eg, '$ env seed=12345 java main'
String seed1 = System.getProperty ("seed", "54321");

String seed2 = System.getenv().getOrDefault("seed","54321"));

long seed = Long.getLong ("seed", System.nanoTime())
```

IntelliJ



Run/Debug Configurations

Name: HelloWorld

☐ Share ☐ Single instance only

- Application
  - HelloWorld
- Defaults
  - Ant Target
  - Applet
  - Application
  - Compound
  - Gradle
  - Griffon
  - Groovy
  - JAR Application
  - Java Scratch
  - JUnit
  - Kotlin
  - Kotlin script
  - Maven
  - Plugin
  - Remote
  - TestNG
  - XSLT

Configuration | Code Coverage | Logs

Main class: ɔn.HelloWorld

VM options: `-Djavax.xml.accessExternalSchema=http`

Program arguments:

Working directory:

Environment variables:

Use classpath of module:

☐ Include dependencies with "Provided" scope

JRE: Default (1.8 - SDK of 'AuthZForceCeCorePdpEngineTestImplementation' module)

Shorten command line: user-local default: none - java [options] classname [args]

☐ Enable capturing form snapshots

Before launch: Build, Activate tool window

Build

☐ Show this page ☑ Activate tool window

OK | Cancel | Apply

Comand Line

```
java  -Dseed=345 -Xmx2g   MyClassName   arg1 "arg with spaces" arg3
```

The executable program `java` needs to be in the command-lines shell's path.
There needs to be an entry point in the java class `MyClassName`.