```java
void main() {
    IO.println("Hello, world!");
}
```

Introduction to Java

Java Fundamentals

Agenda

- Java Background
  - What Is Java?
  - Compilation
  - History
  - Design Principles
- Java Design
  - Language Type
  - Why Static Types?
- Java Project Structure

Java Background

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Language that takes the code you write and translates (compiles) it into different code

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Language that takes the code you write and translates (compiles) it into different code

- Written code is human-readable and information-dense

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Language that takes the code you write and translates (compiles) it into different code

- Written code is human-readable and information-dense
- Compiled code is super efficient due to compiler optimizations

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Language that takes the code you write and translates (compiles) it into different code

- Written code is human-readable and information-dense
- Compiled code is super efficient due to compiler optimizations
- Compiled code is typically not human-readable

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

- Written code is compiled into bytecode and run on a virtual machine

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

- Written code is compiled into bytecode and run on a virtual machine
- Bytecode is compiled into machine code as needed

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

- Written code is compiled into bytecode and run on a virtual machine
- Bytecode is compiled into machine code as needed
- Slower than code that is compiled directly to machine code

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

- Written code is compiled into bytecode and run on a virtual machine
- Bytecode is compiled into machine code as needed
- Slower than code that is compiled directly to machine code
- System independent, and code can be analyzed and changed at runtime

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Variables have fixed data types which must be known at compile-time

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Create many types of applications

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Create many types of applications

- Object-oriented programming (OOP) (mainly)

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Create many types of applications

- Object-oriented programming (OOP) (mainly)
- Imperative programming

What Is Java?

Java is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Create many types of applications

- Object-oriented programming (OOP) (mainly)
- Imperative programming
- Some functional-programming features

Java Terminology

Java Terminology

- *Java bytecode* is what compiled Java code is called

Java Terminology

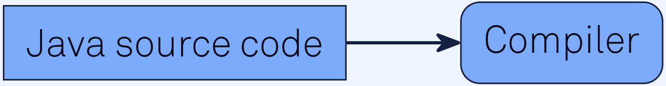
- *Java bytecode* is what compiled Java code is called
- The virtual machine used to execute Java applications is known as the *Java virtual machine (JVM)*

Java Compilation

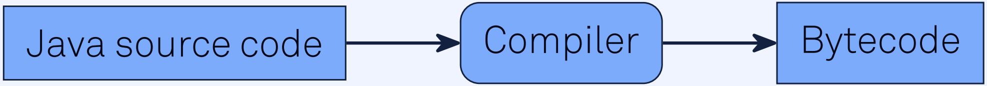
Java Compilation

Java source code

Java Compilation



Java Compilation

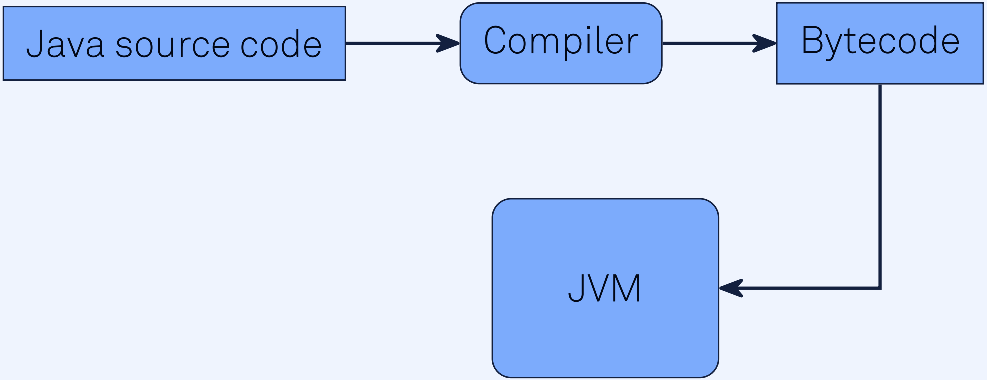


Java source code

Compiler

Bytecode

Java Compilation

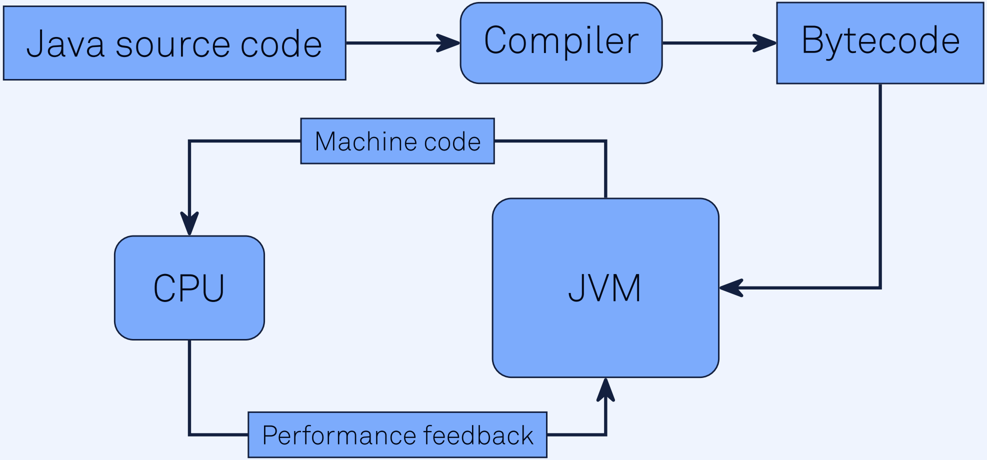

Java source code

Compiler

Bytecode

JVM

Java Compilation



Java source code

Compiler

Bytecode

Machine code

CPU

JVM

Performance feedback

A Brief History

A Brief History

- Developed by James Gosling at Sun Microsystems from 1991 to 1995 with the help of Mike Sheridan and Patrick Naughton

A Brief History

- Developed by James Gosling at Sun Microsystems from 1991 to 1995 with the help of Mike Sheridan and Patrick Naughton
- Released in 1995

A Brief History

- Developed by James Gosling at Sun Microsystems from 1991 to 1995 with the help of Mike Sheridan and Patrick Naughton
- Released in 1995
- Follows C-style syntax which would be familiar to most programmers at the time

A Brief History

- Developed by James Gosling at Sun Microsystems from 1991 to 1995 with the help of Mike Sheridan and Patrick Naughton
- Released in 1995
- Follows C-style syntax which would be familiar to most programmers at the time
- Java has not been officially standardized by any standardization committee

A Brief History

- Developed by James Gosling at Sun Microsystems from 1991 to 1995 with the help of Mike Sheridan and Patrick Naughton
- Released in 1995
- Follows C-style syntax which would be familiar to most programmers at the time
- Java has not been officially standardized by any standardization committee
- C# (Microsoft) was heavily based on Java, but it is now significantly different

Design Principles

Design Principles

- It must be simple, object-oriented, and familiar

Design Principles

- It must be simple, object-oriented, and familiar
- It must be robust and secure

Design Principles

- It must be simple, object-oriented, and familiar
- It must be robust and secure
- It must be architecture-neutral and portable

Design Principles

- It must be simple, object-oriented, and familiar
- It must be robust and secure
- It must be architecture-neutral and portable
- It must execute with high performance

Design Principles

- It must be simple, object-oriented, and familiar
- It must be robust and secure
- It must be architecture-neutral and portable
- It must execute with high performance
- It must be interpreted, threaded, and dynamic

Java Design

What Type of Language Is Java

What Type of Language Is Java

- Java is an application programming language

What Type of Language Is Java

- Java is an application programming language
    - Console apps

What Type of Language Is Java

- Java is an application programming language
    - Console apps
    - Games (Swing, JavaFX, various engines)

What Type of Language Is Java

- Java is an application programming language
  - Console apps
  - Games (Swing, JavaFX, various engines)
  - Web apps (Spring)

What Type of Language Is Java

- Java is an application programming language
  - Console apps
  - Games (Swing, JavaFX, various engines)
  - Web apps (Spring)
  - Desktop and mobile apps (Swing, JavaFX, Android)

What Type of Language Is Java

- Java is an application programming language
    - Console apps
    - Games (Swing, JavaFX, various engines)
    - Web apps (Spring)
    - Desktop and mobile apps (Swing, JavaFX, Android)
- It’s a *high-level* language compared to C and a *very high-level* language compared to Assembly language

Language-level Breakdown

Language-level Breakdown

- Relative to Assembly language

Language-level Breakdown

- Relative to Assembly language
  - Low level (machine code, Assembly)

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)
        - Hard to read

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)
        - Hard to read
    - High level (BASIC, C, C++, COBOL, FORTRAN)

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)
        - Hard to read
    - High level (BASIC, C, C++, COBOL, FORTRAN)
        - Add abstractions

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)
        - Hard to read
    - High level (BASIC, C, C++, COBOL, FORTRAN)
        - Add abstractions
        - Automate some control over hardware (stack)

Language-level Breakdown

- Relative to Assembly language
  - Low level (machine code, Assembly)
    - “Close to the hardware”
    - Specific control over hardware (stack)
    - Hard to read
  - High level (BASIC, C, C++, COBOL, FORTRAN)
    - Add abstractions
    - Automate some control over hardware (stack)
    - Use natural-language elements

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)
        - Hard to read
    - High level (BASIC, C, C++, COBOL, FORTRAN)
        - Add abstractions
        - Automate some control over hardware (stack)
        - Use natural-language elements
    - Very high level (Java, JavaScript, Python, Ruby, C#)

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)
        - Hard to read
    - High level (BASIC, C, C++, COBOL, FORTRAN)
        - Add abstractions
        - Automate some control over hardware (stack)
        - Use natural-language elements
    - Very high level (Java, JavaScript, Python, Ruby, C#)
        - Add many abstractions

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)
        - Hard to read
    - High level (BASIC, C, C++, COBOL, FORTRAN)
        - Add abstractions
        - Automate some control over hardware (stack)
        - Use natural-language elements
    - Very high level (Java, JavaScript, Python, Ruby, C#)
        - Add many abstractions
        - Remove details about hardware control

Language-level Breakdown

Language-level Breakdown

- Relative to C

Language-level Breakdown

- Relative to C
    - Low level (C)

Language-level Breakdown

- Relative to C
    - Low level (C)
        - A lot of hardware control

Language-level Breakdown

- Relative to C
    - Low level (C)
        - A lot of hardware control
        - Not many abstractions

Language-level Breakdown

- Relative to C
    - Low level (C)
        - A lot of hardware control
        - Not many abstractions
    - Middle level (C++)

Language-level Breakdown

- Relative to C
    - Low level (C)
        - A lot of hardware control
        - Not many abstractions
    - Middle level (C++)
        - A blend of abstractions and hardware control

Language-level Breakdown

- Relative to C
    - Low level (C)
        - A lot of hardware control
        - Not many abstractions
    - Middle level (C++)
        - A blend of abstractions and hardware control
        - “Smart” memory management

Language-level Breakdown

- Relative to C
    - Low level (C)
        - A lot of hardware control
        - Not many abstractions
    - Middle level (C++)
        - A blend of abstractions and hardware control
        - “Smart” memory management
    - High level (Java, Python, C#)

Language-level Breakdown

- Relative to C
    - Low level (C)
        - A lot of hardware control
        - Not many abstractions
    - Middle level (C++)
        - A blend of abstractions and hardware control
        - “Smart” memory management
    - High level (Java, Python, C#)
        - Many abstractions

Language-level Breakdown

- Relative to C
  - Low level (C)
    - A lot of hardware control
    - Not many abstractions
  - Middle level (C++)
    - A blend of abstractions and hardware control
    - “Smart” memory management
  - High level (Java, Python, C#)
    - Many abstractions
    - Automated memory management (garbage collector)

Why Static Types?

Why Static Types?

- Data types are important to programmers

Why Static Types?

- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime

Why Static Types?

- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime
- Meaning of code is clearer

Why Static Types?

- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime
- Meaning of code is clearer
- What do Python and JavaScript do?

Why Static Types?

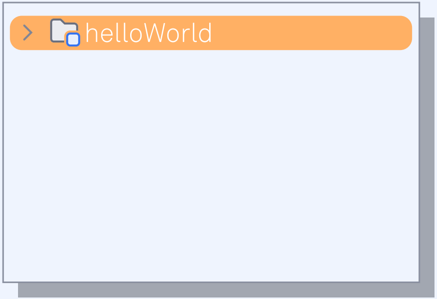
- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime
- Meaning of code is clearer
- What do Python and JavaScript do?
    - mypy

Why Static Types?

- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime
- Meaning of code is clearer
- What do Python and JavaScript do?
    - mypy
    - TypeScript/JSDoc comments

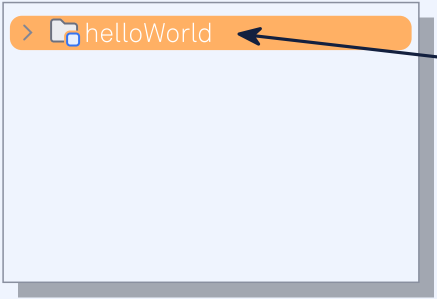
Java Project Structure

Project Structure



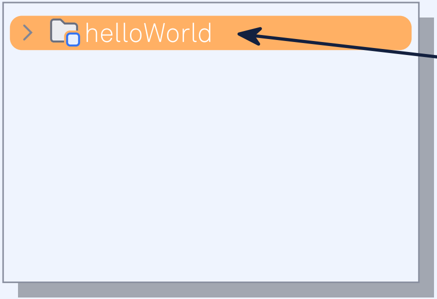
helloWorld

Project Structure



Java project directory

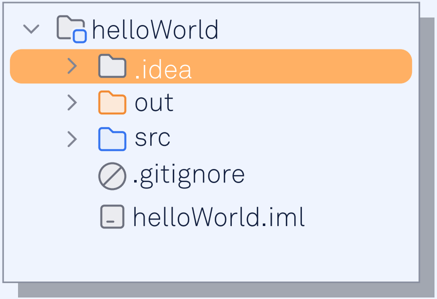
Project Structure



Java project directory

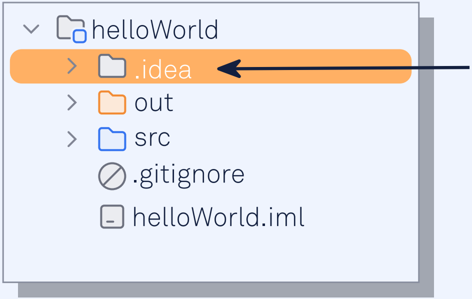
- This directory holds everything for a Java project

Project Structure



```
helloWorld
  .idea
  out
  src
  .gitignore
  helloWorld.iml
```

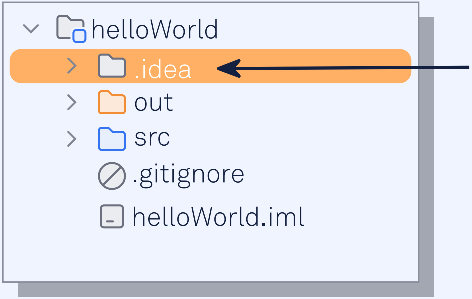
Project Structure



```
helloWorld
  .idea
  out
  src
  .gitignore
  helloWorld.iml
```

IntelliJ settings directory

Project Structure

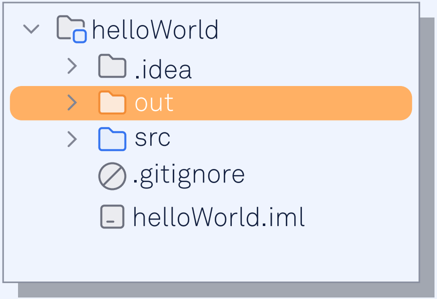


```
helloWorld
  .idea
  out
  src
  .gitignore
  helloWorld.iml
```

IntelliJ settings directory

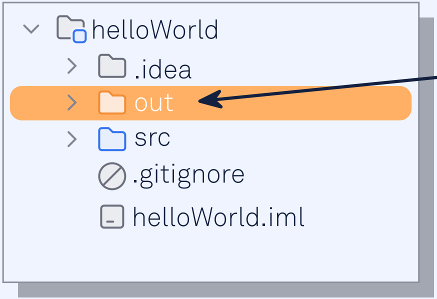
- This file holds IDE settings.

Project Structure



- helloWorld
  - .idea
  - out
  - src
  - .gitignore
  - helloWorld.iml

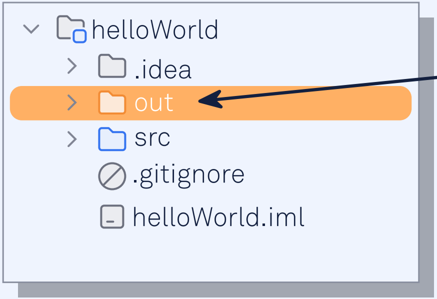
Project Structure



```
helloWorld
  .idea
  out
  src
  .gitignore
  helloWorld.iml
```

Output directory

Project Structure

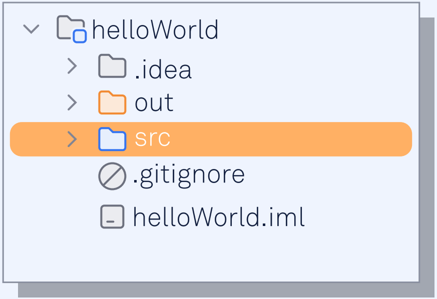


```
helloWorld
  .idea
  out
  src
  .gitignore
  helloWorld.iml
```

Output directory

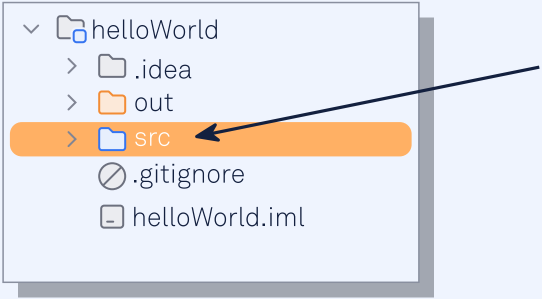
- This is where the bytecode generated by the compiler goes

Project Structure



```
helloWorld
    .idea
    out
    src
    .gitignore
    helloWorld.iml
```

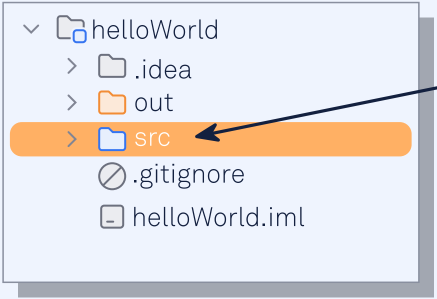
Project Structure



helloWorld
- .idea
- out
- src
- .gitignore
- helloWorld.iml

Source directory

Project Structure

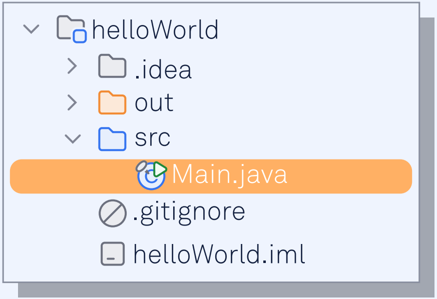


```
helloWorld
  .idea
  out
  src
  .gitignore
  helloWorld.iml
```

Source directory

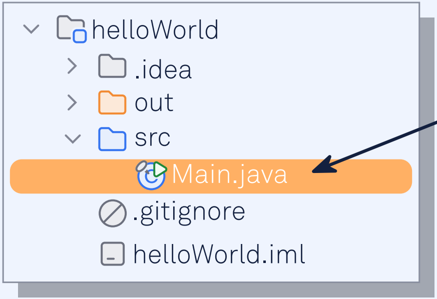
- This is where the Java code goes

Project Structure



```
helloWorld
    .idea
    out
    src
        Main.java
    .gitignore
    helloWorld.iml
```

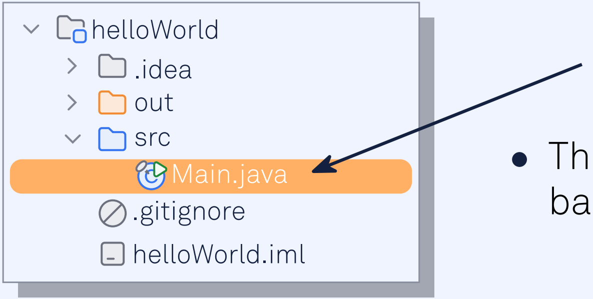
Project Structure



```
helloWorld
  .idea
  out
  src
    Main.java
  .gitignore
  helloWorld.iml
```

Java `Main` class

Project Structure

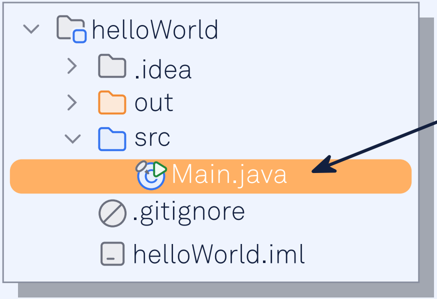


```
helloWorld
  .idea
  out
  src
    Main.java
  .gitignore
  helloWorld.iml
```

Java `Main` class

- This is the main class for a basic console application

Project Structure

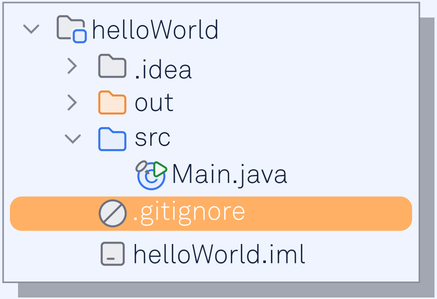


```
helloWorld
  .idea
  out
  src
    Main.java
  .gitignore
  helloWorld.iml
```

Java **`Main`** class

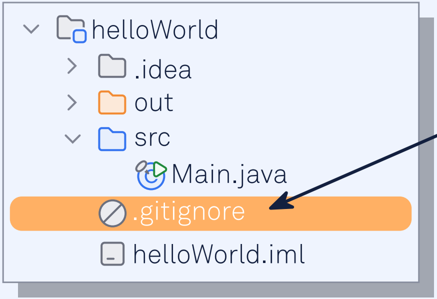
- This is the main class for a basic console application
- This is where the code from the title slide goes

Project Structure



```
helloWorld
    .idea
    out
    src
        Main.java
    .gitignore
    helloWorld.iml
```

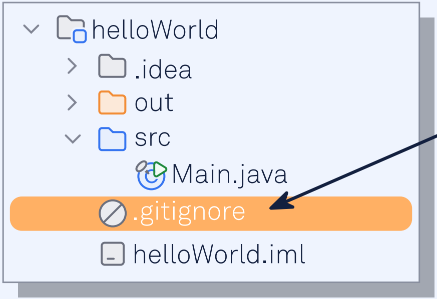
Project Structure



```
helloWorld
  .idea
  out
  src
    Main.java
  .gitignore
  helloWorld.iml
```

Git ignore file

Project Structure

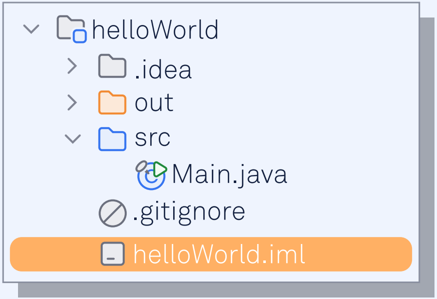


```
helloWorld
    .idea
    out
    src
        Main.java
    .gitignore
    helloWorld.iml
```

Git ignore file

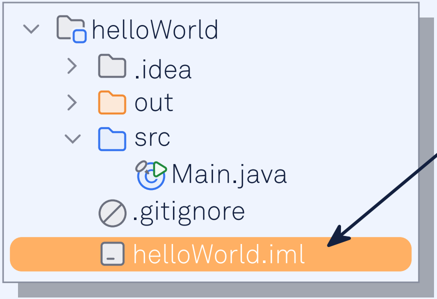
- This file is used when uploading to a remote repository

Project Structure



```
helloWorld
    .idea
    out
    src
        Main.java
    .gitignore
    helloWorld.iml
```

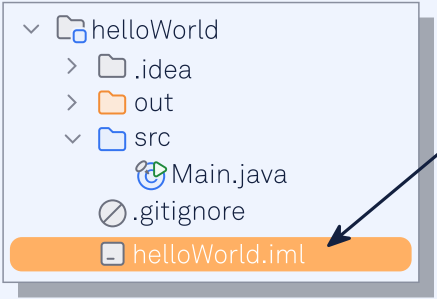
Project Structure



```
helloWorld
  .idea
  out
  src
    Main.java
  .gitignore
  helloWorld.iml
```

Idea module file

Project Structure



```
helloWorld
    .idea
    out
    src
        Main.java
    .gitignore
    helloWorld.iml
```

Idea module file

- This file manages more IDE settings

Any Questions?