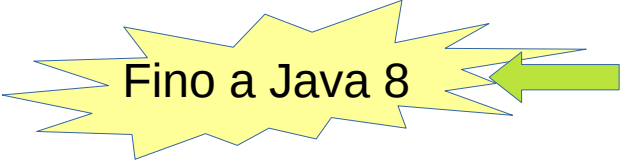


Java SE: Eclipse

- Introduzione a Java
- Eclipse
 - Maven
 - Git
- Jenkins
 - Git – Maven
- Progetto di riferimento
 - <https://github.com/egalli64/mpjp> (*modulo 1*)

Java

Linguaggio di programmazione static-typed, general-purpose, imperativo, object-oriented, class-based, multi-platform, network-centric progettato da James Gosling @ Sun Microsystems.



Fino a Java 8

- **JVM**: Java Virtual Machine
- JRE: Java Runtime Environment
- **JDK**: Java Development Kit

Versioni

- 23 maggio 1995: prima release
- 1998 1.2 (J2SE)
- 2004 1.5 (J2SE 5.0)
- 2011 Java SE 7
- 03/2014 Java SE 8 (LTS)
- 09/2018 Java SE 11 (LTS)
- 09/2020 Java SE 15



SE: Standard Edition

EE: Enterprise Edition

LTS: Long-Term Support

Link utili

Java Language Specifications: <https://docs.oracle.com/javase/specs/>

Java SE Documentation: <https://docs.oracle.com/en/java/javase/index.html>

Java SE 8 API Specification: <https://docs.oracle.com/javase/8/docs/api/index.html>

The Java Tutorials (JDK 8): <https://docs.oracle.com/javase/tutorial/>

Java SE Downloads

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>

<https://adoptopenjdk.net/>

Hello Java a riga di comando

- Generazione del **bytecode**

`javac Hello.java`

bin di jdk non nel system path!
vedi: impostazioni di Windows
path → variabili d'ambiente di sistema

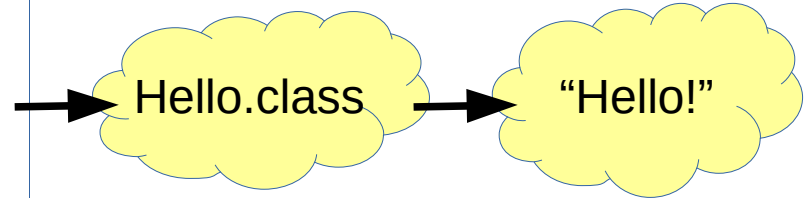
- *(Per i curiosi)* Visualizzazione del bytecode disassemblato

`javap -c Hello`

- Generazione del **codice macchina** ed **esecuzione**

`java Hello`

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



Integrated Development Environment

- Semplifica lo sviluppo di applicazioni
 - IntelliJ IDEA
 - Eclipse IDE
 - Installer: <https://www.eclipse.org/downloads/>
 - vedi anche STS – Spring Tool Suite <https://spring.io/tools>
 - Apache NetBeans
 - ...
- Alternative più leggere:
 - Microsoft VS Code (“code editor”)
 - ...

Build automation con Maven

- Build automation
 - Compilazione del codice sorgente
 - Packaging dell'eseguibile
 - Esecuzione automatica dei test
- UNIX make, Ant, Maven, Gradle
- **Apache Maven**, supportato da tutti i principali IDE per Java
 - <https://maven.apache.org/>
 - **pom.xml** (POM: Project Object Model)
 - I processi seguono convenzioni stabilite, solo le eccezioni vanno indicate
 - Le dipendenze implicano il download automatico delle librerie richieste

Nuovo progetto Maven in Eclipse

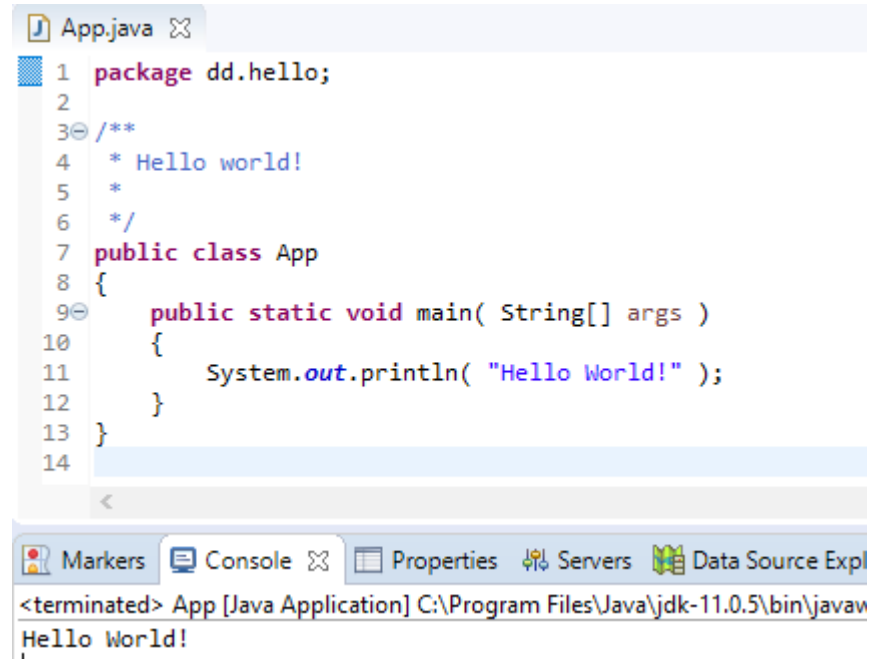
- Creare un progetto Maven
 - File, New, Maven Project
 - È necessario specificare solo **group id** e **artifact id**
 - Il progetto risultante è per Java 5
- Nel POM specifichiamo le nostre **variazioni** (vedi slide successive)
 - Properties
 - Dependencies
- A volte occorre forzare l'update del progetto dopo aver cambiato il POM
 - Alt-F5 (o right-click sul nome del progetto → Maven, Update project)

Properties

- Definizione di costanti relative al POM
- Ad esempio:
 - Codifica nel codice sorgente
 - Versione di Java (source e target)

```
<properties>  
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
  <maven.compiler.source>11</maven.compiler.source>  
  <maven.compiler.target>11</maven.compiler.target>  
</properties>
```

Hello World!



The screenshot shows an IDE window with a file named 'App.java'. The code is as follows:

```
1 package dd.hello;
2
3 /**
4  * Hello world!
5  *
6  */
7 public class App
8 {
9     public static void main( String[] args )
10    {
11        System.out.println( "Hello World!" );
12    }
13 }
14
```

Below the code editor, the 'Console' tab is active, displaying the output of the program:

```
<terminated> App [Java Application] C:\Program Files\Java\jdk-11.0.5\bin\javaw
Hello World!
```

Aggiungere una dependency

- Ricerca su repository Maven (central e altri)
 - <https://search.maven.org/>,
<https://mvnrepository.com/>
- Ad esempio:
 - JUnit (4.13), JUnit Jupiter engine (5.6.2)

```
<dependency>  
  <groupId>junit</groupId>  
  <artifactId>junit</artifactId>  
  <version>4.13</version>  
</dependency>
```

```
<dependency>  
  <groupId>org.junit.jupiter</groupId>  
  <artifactId>junit-jupiter-engine</artifactId>  
  <version>5.6.2</version>  
</dependency>
```

Tra le `<dependencies>`

Vogliamo usare Junit solo in test, perciò aggiungiamo:
`<scope>test</scope>`

Import di un progetto via Git

- Da Eclipse
 - File, Import ..., Git, Projects from Git (with smart import)
 - Clone URI
 - Fornita da GitHub.
 - Il progetto di riferimento è: <https://github.com/egalli64/mpjp>
 - Se Eclipse non lo riconosce come progetto Maven, va importato come “General Project” e poi “mavenizzato”
 - Oppure, se il repository è già stato clonato
 - import del progetto come Existing local repository

Nuovo repository Git in Eclipse

- GitHub, creazione di un nuovo repository “xyz”
- Shell di Git, nella directory git locale:

```
git clone <url di xyz.git>
```

(oppure si può clonarlo dalla prospettiva Git di Eclipse)

- Eclipse: creazione di un nuovo progetto
 - Location: directory del repository appena clonato git/xyz
- Il nuovo progetto viene automaticamente collegato da Eclipse al repository Git presente nel folder

Team per Git in Eclipse

- Right click sul nome del progetto, Team
 - Pull (o Pull... per il branch corrente)
 - Commit rimanda alla view “Git staging”
 - Push to upstream (per il branch corrente)
 - Switch To, New branch...
 - Basta specificare il nome del nuovo branch
 - Switch To, per cambiare il branch corrente
 - Merge branch, per fondere due branch

.gitignore in Eclipse

- Per ignorare file o folder
 - Come già visto, file .gitignore
 - Oppure: right-click sulla risorsa, Team, Ignore
- Eclipse annota le icone di file e folder con simboli per mostrare come sono gestiti da Git
 - punto di domanda: risorsa sconosciuta
 - asterisco: risorsa staged per commit
 - più: risorsa aggiunta a Git ma non ancora tracked
 - assenza di annotazioni: risorsa ignorata

Jenkins Maven Git

- Progetto Java Maven su GitHub
- Jenkins con plugin
 - Git: <https://plugins.jenkins.io/git/> e Maven: <https://plugins.jenkins.io/maven-plugin/>
 - Jenkins deve sapere dove sono i tool sulla nostra macchina: `/configureTools/`
- New Jenkins Item “simple” come Maven project
- Configurazione `/job/simple/configure`
 - Source Code Management: <https://github.com/egalli64/simple.git>
 - Additional Behaviours: *Clean before checkout*, ...
 - Build:
 - Root POM: pom.xml
 - Goals and options: package [...]