Lesson 12

18.01.2024

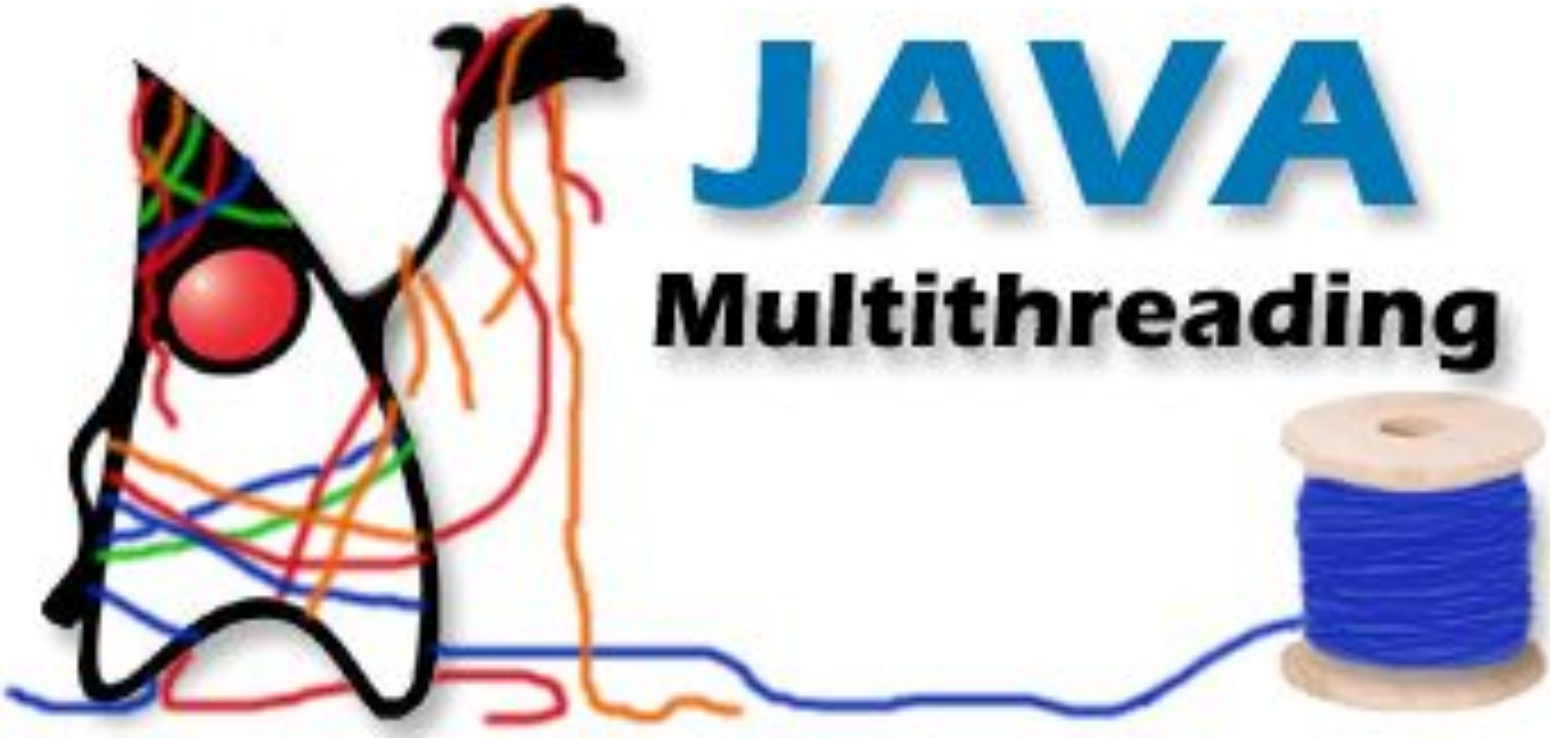
```
public class Ex1 {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("abc");
        String s = "abc";
        sb.reverse().append("d");
        s.toUpperCase().concat("d");
        System.out.println("." + sb + ". ." + s + ".");
    }
}
```

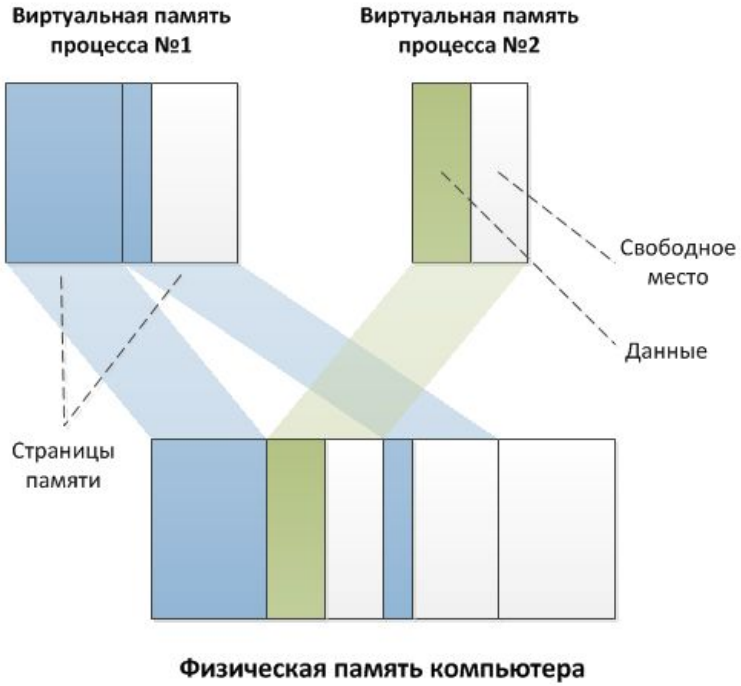
```
public class Ex2 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("apple");
        list.add("carrot");
        list.add("banana");
        list.add(1, "plum");
        System.out.println(list);
    }
}
```

```
public class Ex3 {
    public static void main(String[] args) {
        String s = "JAVA";
        s = s + "rock";
        s = s.substring(4, 8);
        s.toUpperCase();
        System.out.println(s);
    }
}
```

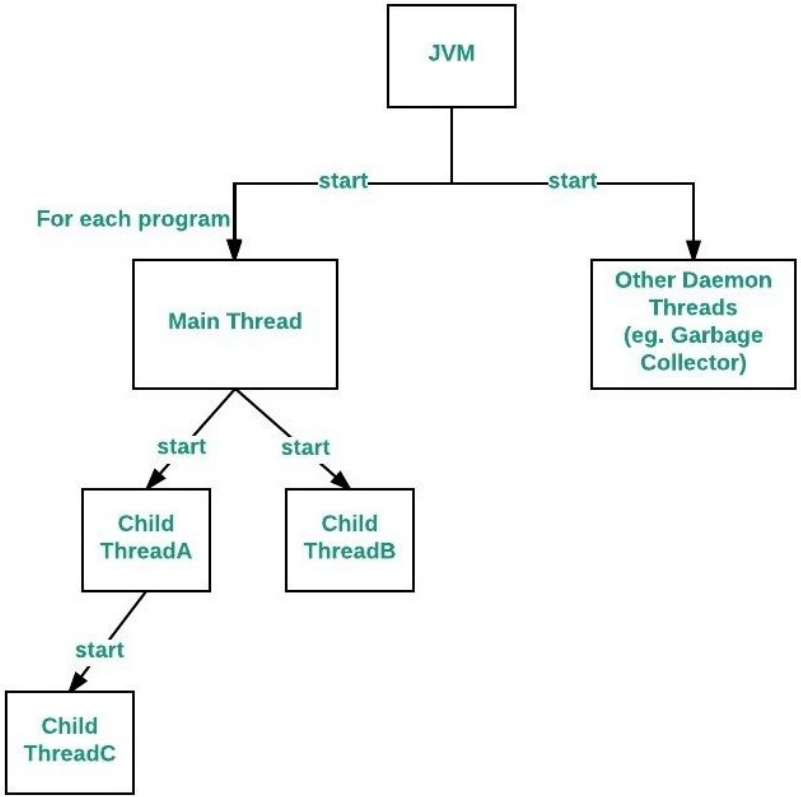
```
public class Ex4 {
    public static void main(String[] args) {
        String[] name = {"Sasha", "Ivan", "Masha"};
        List<String> names = name.asList();
        names.set(0, "Kate");
        System.out.println(name[0]);
    }
}
```

```
public class Ex5 {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("0123456789");
        sb.delete(2, 8);
        sb.append("-").insert(2, "+");
        System.out.println(sb);
    }
}
```

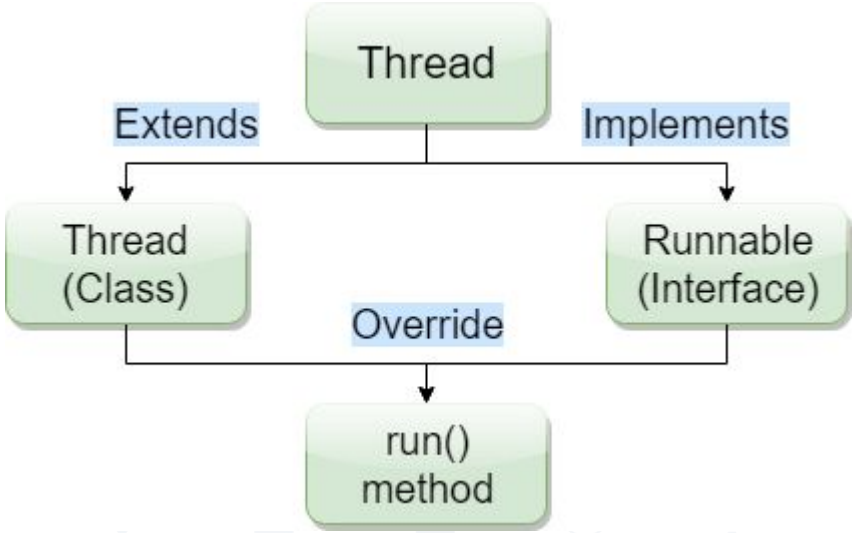

JAVA
Multithreading



Процес - це сукупність коду та даних, що розділяють загальний віртуальний адресний простір. Найчастіше одна програма складається з одного процесу, але бувають і винятки (наприклад, браузер Chrome створює окремий процес для кожної вкладки, що дає йому деякі переваги на кшталт незалежності вкладок один від одного). Процеси ізольовані друг від друга, тому прямий доступом до пам'яті чужого процесу неможливий (взаємодія між процесами здійснюється з допомогою спеціальних засобів).



Потік – це одна одиниця виконання коду. Кожен потік послідовно виконує інструкції процесу, якому належить, паралельно виконується з іншими потоками цього процесу.



Зауважте, що обидва приклади викликають метод Thread.start() для запуску нового потоку. Саме він запускає окремий потік. Якщо просто викликати метод run(), код буде виконуватися в тому ж потоці, окремий потік створюватися не буде.

Метод **sleep** класу Thread зупиняє виконання поточного потоку на вказаний час. Він використовується, коли потрібно звільнити процесор, щоб він зайнявся іншими потоками чи процесами, або завдання інтервалу між якими-небудь діями.

Переривання (**interrupt**) - це сигнал для потоку, що він повинен припинити робити те, що робить зараз, і робити щось інше. Що має робити потік у відповідь переривання, вирішує програміст, але зазвичай потік завершується.
Потік відправляє переривання, викликаючи метод public void interrupt() класу Thread. Для того щоб механізм переривання працював коректно, потік, що переривається, повинен підтримувати можливість переривання своєї роботи.

Метод **join** дозволяє одному потоку чекати завершення іншого потоку. Якщо t є екземпляром класу Thread, чий потік на даний момент продовжує виконуватись, то

t.join();

призведе до призупинення виконання поточного потоку, поки потік t не завершить свою роботу.

Як і методи sleep, методи join відповідають на сигнал переривання, зупиняючи процес очікування і кидаючи виняток InterruptedException.

Статичний метод **Thread.yield()** змушує процесор перейти на обробку інших потоків системи. Метод може бути корисним, наприклад, коли потік очікує настання якоїсь події і необхідно, щоб перевірка його наступу відбувалася якомога частіше. У цьому випадку можна помістити перевірку події та метод.

Деякі корисні методи класу Thread

boolean isAlive() — повертає true, якщо myThready() виконується і false, якщо потік ще не був запущений або був завершений.
setName(String threadName) - Вказує ім'я потоку.
String getName() – Отримує ім'я потоку. Ім'я потоку – асоційований з ним рядок, який у деяких випадках допомагає зрозуміти, який потік виконує певну дію. Іноді це корисно.
static Thread Thread.currentThread() - статичний метод, що повертає об'єкт потоку, в якому він був викликаний.
long getId() – повертає ідентифікатор потоку. Ідентифікатор – унікальне число, яке присвоєно потоку.

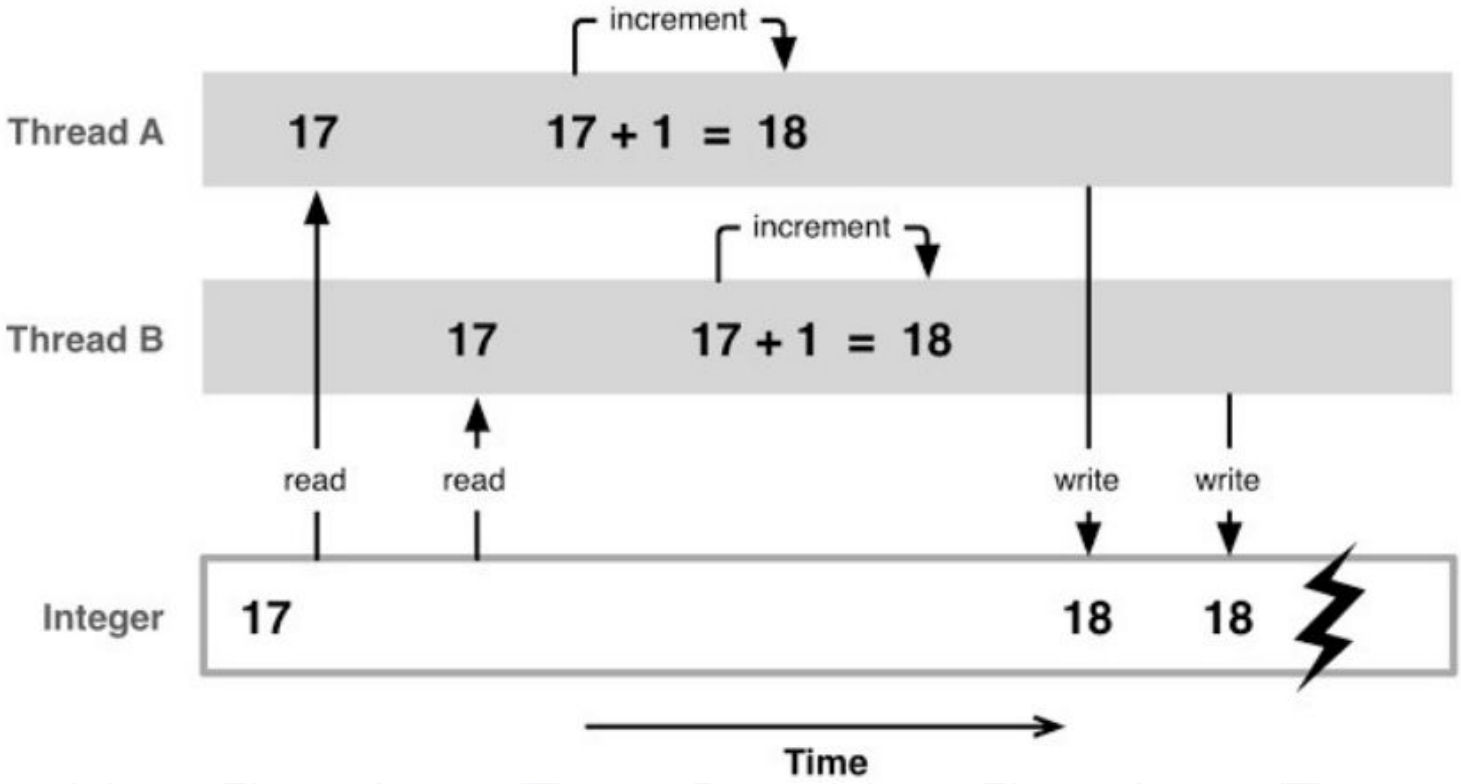
Пріоритети потоків

Кожен потік у системі має свій пріоритет. Пріоритет – це певне число в об'єкті потоку, вище значення якого означає більший пріоритет. Система в першу чергу виконує потоки з більшим пріоритетом, а потоки з меншим пріоритетом отримують процесорний час тільки тоді, коли більш привілейовані побратими простоюють.

Працювати з пріоритетами потоку можна за допомогою двох функцій:

void setPriority(int priority) – встановлює пріоритет потоку.
Можливі значення priority - MIN_PRIORITY, NORM_PRIORITY та MAX_PRIORITY.

int getPriority() – отримує пріоритет потоку.


increment
Thread A
17
17 + 1 = 18
increment
Thread B
17
17 + 1 = 18
read
read
write
write
Integer
17
18
18
Time

Синхронізація потоків

Всі потоки, що належать до одного процесу, поділяють деякі загальні ресурси (адресний простір, відкриті файли). Що станеться, якщо один потік ще не закінчив працювати з якимось загальним ресурсом, а система переключилася на інший потік, який використовує той самий ресурс?

Коли два або більше потоків мають доступ до одного розділеного ресурсу, вони потребують того, що ресурс буде використаний тільки одним потоком одночасно. Процес, за допомогою якого це досягається, називається синхронізацією.

Синхронізувати прикладний код можна двома способами:
За допомогою синхронізованих методів. Метод оголошується за допомогою ключового слова synchronized:

```
public synchronized void someMethod(){}
```

Укласти виклики методів у блок оператора synchronized:

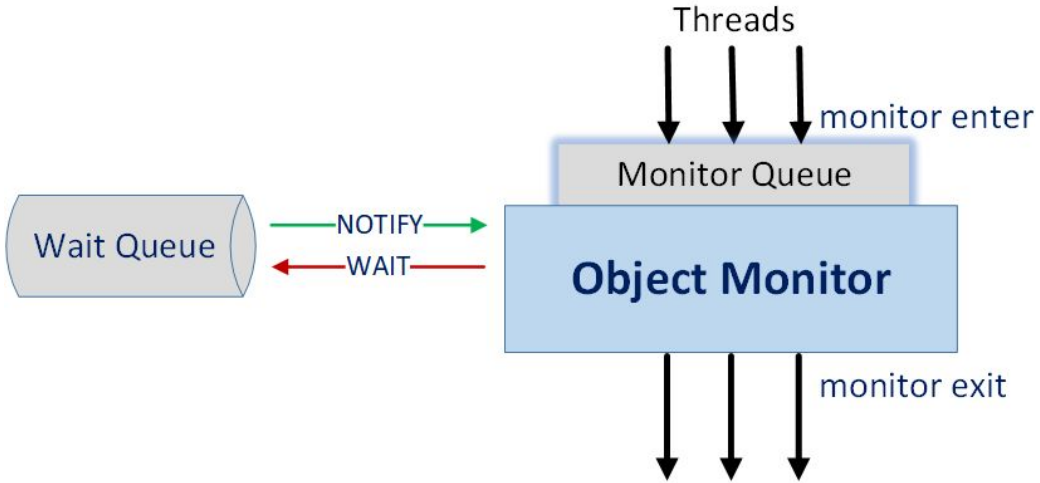
```
synchronized(объект) {
  // операторы, подлежащие синхронизации
}
```

З кожним об’єктом у Java пов’язаний монітор. Монітор - це об'єкт, який використовується як взаємовиключний блок. Коли потік виконання запитує блокування, вони повідомляють, що він включений у монітор.
Тільки один потік виконання може одночасно керувати монітором. Усі інші потоки виконання, які намагаються увійти до заблокованого монітора, будуть призупинені, доки перший потік не вийде з монітора. Кажуть, чекають на монітор.

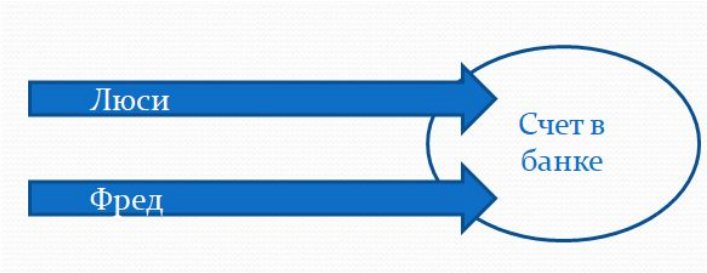
Потік, що керує монітором, може, якщо бажає, повторно ввійти в нього.

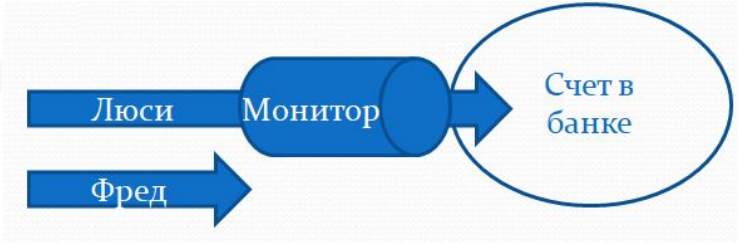
Якщо потік засинає, то він тримає монітор.

Потік може захоплювати кілька моніторів одночасно.



Ми розглянемо різницю між доступом до об’єкта без синхронізації та з синхронізованого коду.





Коли виконання коду досягає синхронізованого оператора, монітор об'єкта блокується, і під час його блокування ексклюзивний доступ до блоку коду має тільки один потік, який створив блок (Lucy).
Після завершення блоку коду монітор об'єкта облікового запису звільняється і стає доступним для інших потоків.
Після звільнення монітора інший потік приймає його, а всі інші потоки продовжують чекати його звільнення.

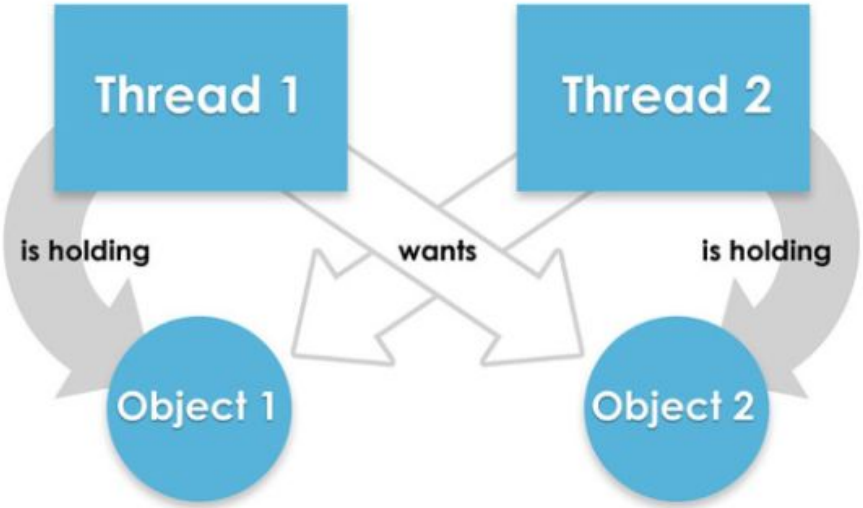
Блокування
Якщо потік намагається увійти в синхронізований метод, а монітор вже зайнятий, то потік блокується на моніторі об'єкта.
Потік потрапляє в спеціальний пул для цього конкретного об'єкта і повинен знаходитися там до моменту звільнення монітора. Після цього потік повертається до працездатного стану.

Deadlock

Необхідно уникати особливого типу помилки, пов’язаної з багатозадачністю, яка називається взаємним блокуванням, яка виникає, коли потоки виконання мають циклічну залежність від пари синхронізованих об’єктів.

Припустимо, один потік виконання потрапляє на монітор об’єкта X, а інший – на монітор об’єкта У. Якщо потік виконання в об’єкті X намагається викликати будь-який синхронізований метод для об’єкта У, він буде заблоковано, як очікувалося.

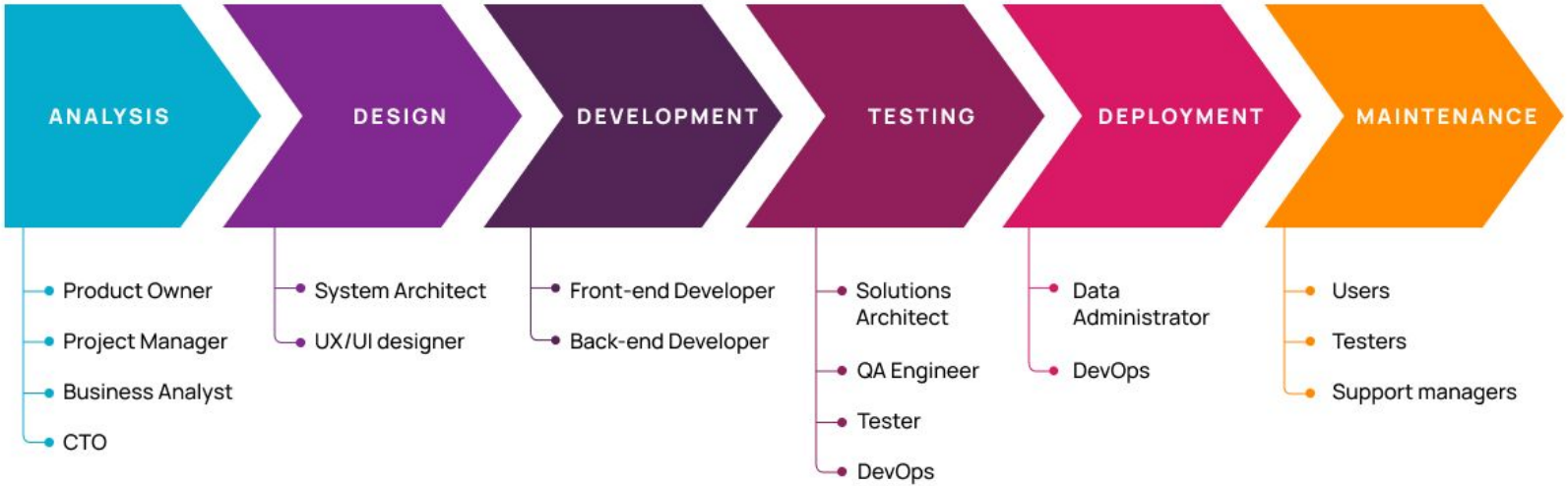
Але якщо потік виконання в об’єкті У, у свою чергу, спробує викликати будь-який синхронізований метод для об’єкта X, то цей потік чекатиме вічно, оскільки для того, щоб отримати доступ до об’єкта X, він повинен зняти блокування з У. так, щоб перший потік виконання міг завершити .



об'єкти **stateless** (з англ. "без стану")
immutable об'єкти (з англ. "неизменяемые")

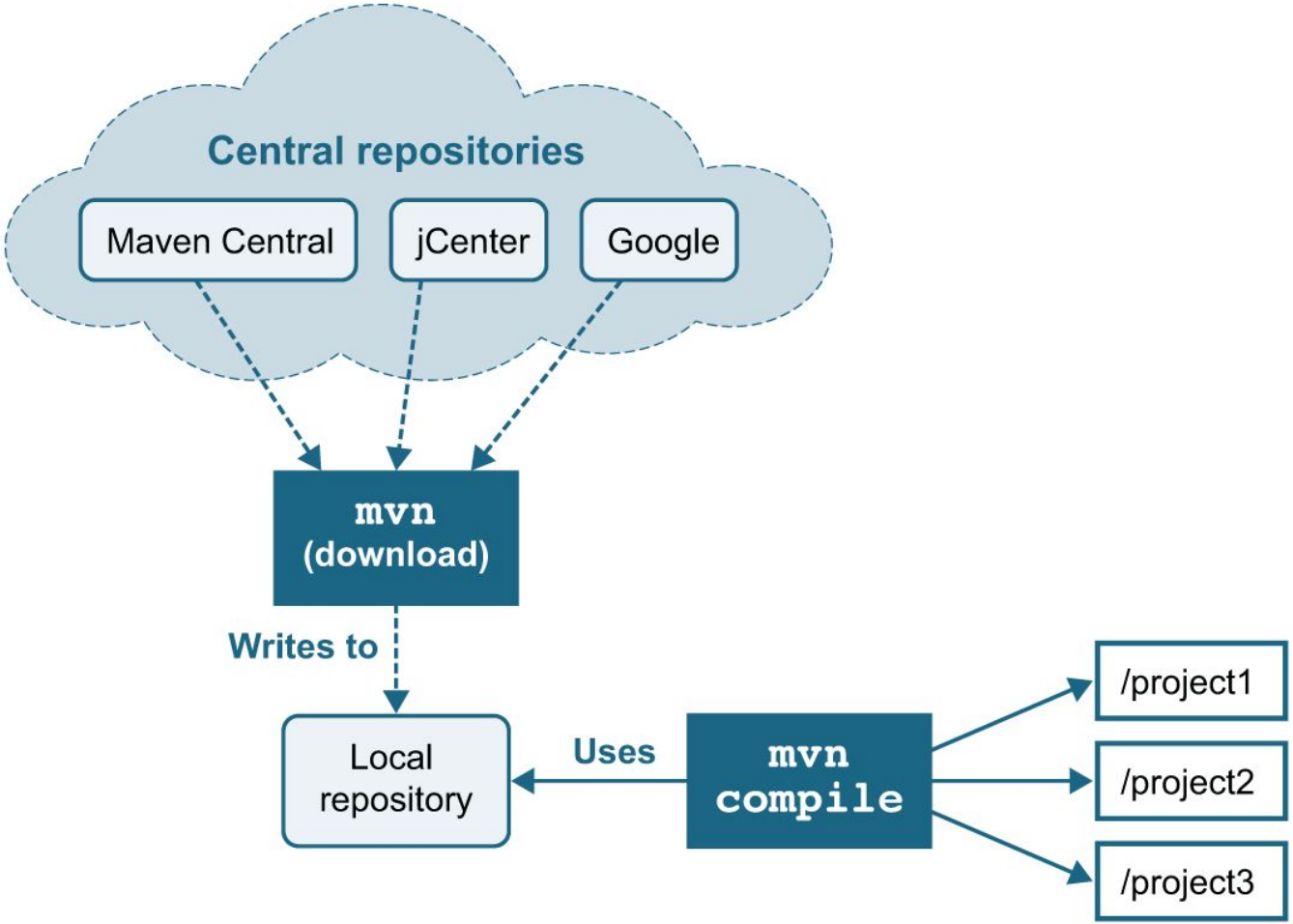
Ці об’єкти використовуються в багатопоточному середовищі, щоб бути впевненими, що ці об’єкти не будуть змінені іншим потоком. Щоб є зафіксували стан раз і назавжди. Інакше довелося б думати про те, як синхронізувати доступ до цих об’єктів з різних потоків. А це сповільнює програму.

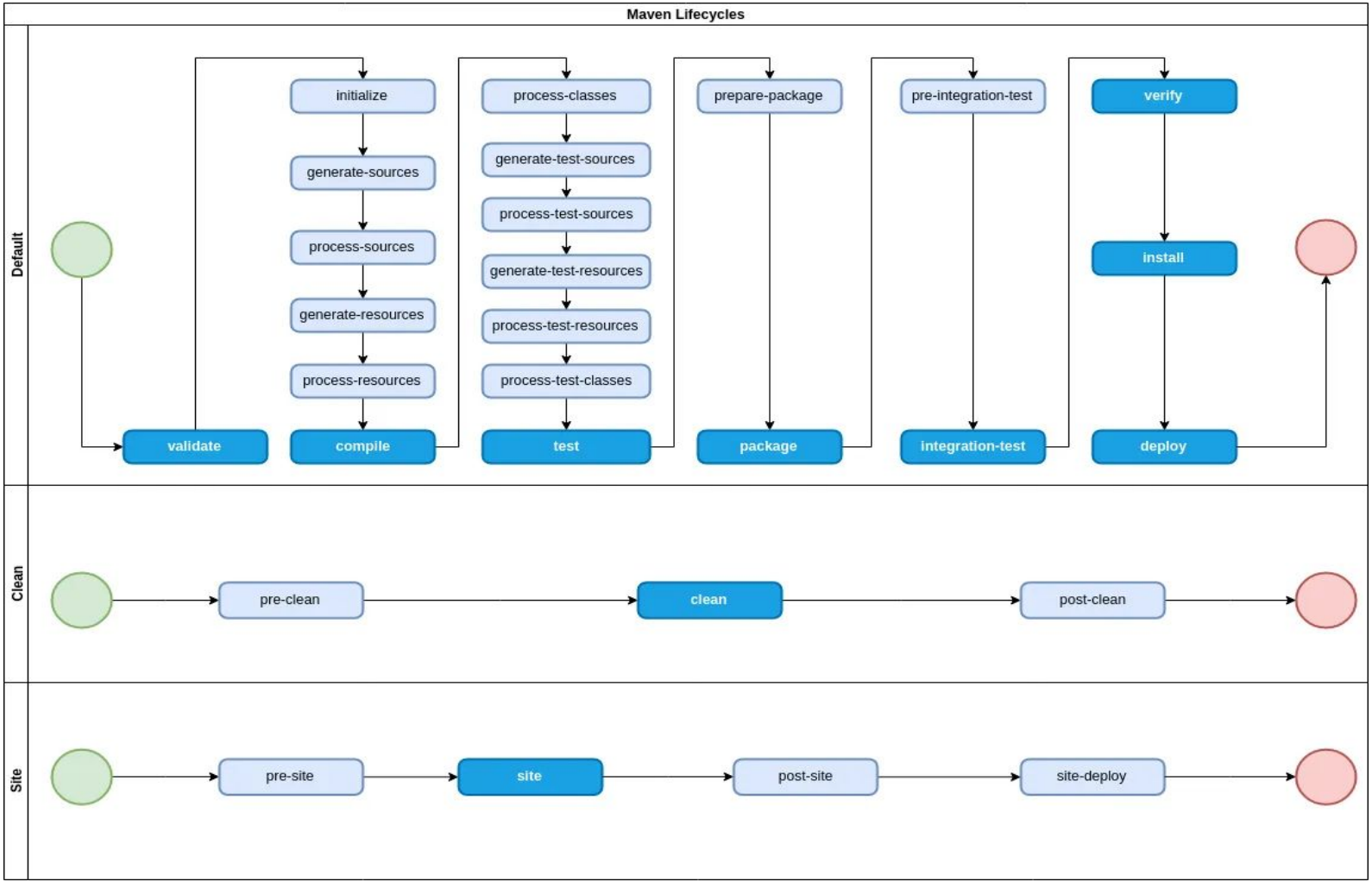
6 Phases of the Software Development Life Cycle



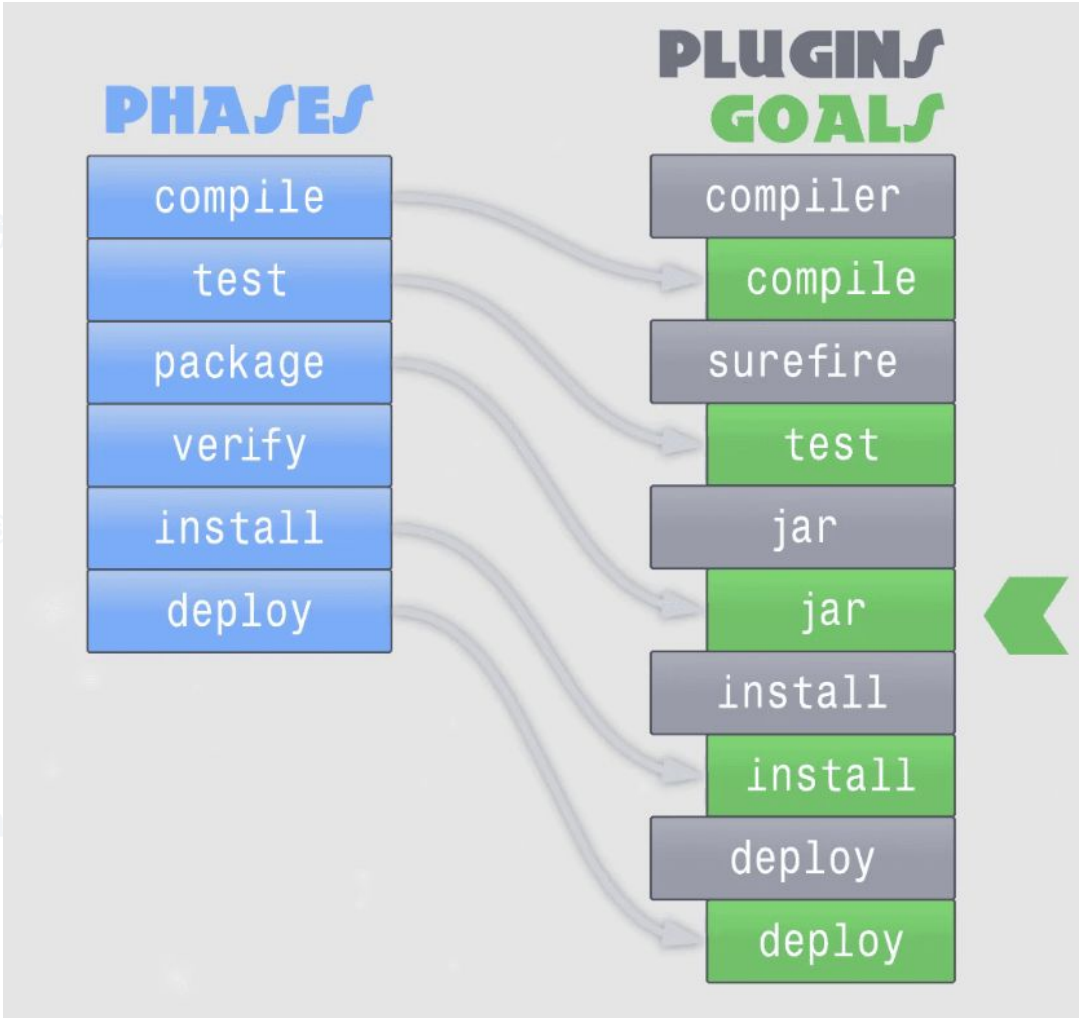

Gradle
Maven™
<APACHE ANT>

- Apache Ant http://ant.apache.org/
- Apache Maven https://maven.apache.org/
- Gradle https://gradle.org/


Central repositories
Maven Central
jCenter
Google
mvn
(download)
Writes to
Local
repository
Uses
mvn
compile
/project1
/project2
/project3


Maven Lifecycles
Default
validate
initialize
generate-sources
process-sources
generate-resources
process-resources
compile
process-classes
generate-test-sources
process-test-sources
generate-test-resources
process-test-resources
process-test-classes
test
prepare-package
package
pre-integration-test
integration-test
verify
install
deploy
Clean
pre-clean
clean
post-clean
Site
pre-site
site
post-site
site-deploy

Default Lifecycle	
Phases	**Description**
process-resources	copy and process the resources into the destination directory, ready for packaging
compile	compile the source code of the project
process-test-resources	copy and process the resources into the destination directory
test-compile	compile the test code into the test destination directory
test	run tests using a suitable unit testing framework.
package	package the build into distributable format, such as a JAR, WAR, or EAR
install	install the package into the local repository, for use as a dependency in other projects locally
deploy	copies the final package to the remote repository for sharing with other developers and projects


PHASES
compile
test
package
verify
install
deploy
PLUGINS
GOALS
compiler
compile
surefire
test
jar
jar
install
install
deploy
deploy

Maven Command	Description
mvn --version	Prints out the version of Maven you are running.
mvn clean	Clears the `target` directory into which Maven normally builds your project.
mvn package	Builds the project and packages the resulting JAR file into the `target` directory.
mvn package -Dmaven.test.skip=true	Builds the project and packages the resulting JAR file into the `target` directory - without running the unit tests during the build.
mvn clean package	Clears the `target` directory and Builds the project and packages the resulting JAR file into the `target` directory.
mvn clean package -Dmaven.test.skip=true	Clears the `target` directory and builds the project and packages the resulting JAR file into the `target` directory - without running the unit tests during the build.
mvn verify	Runs all integration tests found in the project.
mvn clean verify	Cleans the target directory, and runs all integration tests found in the project.
mvn install	Builds the project described by your Maven POM file and installs the resulting artifact (JAR) into your local Maven repository
mvn install -Dmaven.test.skip=true	Builds the project described by your Maven POM file without running unit tests, and installs the resulting artifact (JAR) into your local Maven repository
mvn clean install	Clears the `target` directory and builds the project described by your Maven POM file and installs the resulting artifact (JAR) into your local Maven repository
mvn clean install -Dmaven.test.skip=true	Clears the `target` directory and builds the project described by your Maven POM file without running unit tests, and installs the resulting artifact (JAR) into your local Maven repository
mvn dependency:copy-dependencies	Copies dependencies from remote Maven repositories to your local Maven repository.
mvn clean dependency:copy-dependencies	Cleans project and copies dependencies from remote Maven repositories to your local Maven repository.

mvn dependency:tree	Prints out the dependency tree for your project - based on the dependencies configured in the pom.xml file.
mvn dependency:tree -Dverbose	Prints out the dependency tree for your project - based on the dependencies configured in the pom.xml file. Includes repeated, transitive dependencies.
mvn dependency:tree -Dincludes=com.fasterxml.jackson.core	Prints out the dependencies from your project which depend on the com.fasterxml.jackson.core artifact.
mvn dependency:tree -Dverbose -Dincludes=com.fasterxml.jackson.core	Prints out the dependencies from your project which depend on the com.fasterxml.jackson.core artifact. Includes repeated, transitive dependencies.
mvn dependency:build-classpath	Prints out the classpath needed to run your project (application) based on the dependencies configured in the pom.xml file.

Maven cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Getting started with Maven

Create Java project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
```

Create web project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
-DarchetypeArtifactId=maven-archetype-webapp
```

Create archetype from existing project

```
mvn archetype:create-from-project
```

Main phases

clean — delete target directory
validate — validate, if the project is correct
compile — compile source code, classes stored in target/classes
test — run tests
package — take the compiled code and package it in its distributable format, e.g. JAR, WAR
verify — run any checks to verify the package is valid and meets quality criteria
install — install the package into the local repository
deploy — copies the final package to the remote repository

Useful command line options

-DskipTests=true compiles the tests, but skips running them

-Dmaven.test.skip=true skips compiling the tests and does not run them

-T - number of threads:
-T 4 is a decent default
-T 2C - 2 threads per CPU

-rf, --resume-from resume build from the specified project

-pl, --projects makes Maven build only specified modules and not the whole project

-am, --also-make makes Maven figure out what modules out target depends on and build them too

-o, --offline work offline

-X, --debug enable debug output

-P, --activate-profiles comma-delimited list of profiles to activate

-U, --update-snapshots forces a check for updated dependencies on remote repositories

-ff, --fail-fast stop at first failure

Essential plugins

Help plugin — used to get relative information about a project or the system.
mvn help:describe describes the attributes of a plugin
mvn help:effective-pom displays the effective POM as an XML for the current build, with the active profiles factored in.

Dependency plugin — provides the capability to manipulate artifacts.
mvn dependency:analyze analyzes the dependencies of this project
mvn dependency:tree prints a tree of dependencies

Compiler plugin — compiles your java code.
Set language level with the following configuration:

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</
  artifactId>
 <version>3.6.1</version>
 <configuration>
   <source>1.8</source>
   <target>1.8</target>
 </configuration>
</plugin>
```

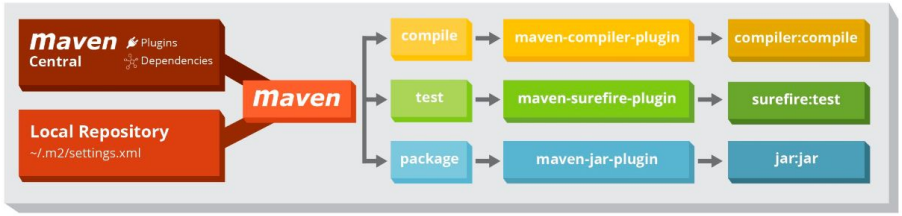
Version plugin — used when you want to manage the versions of artifacts in a project's POM.

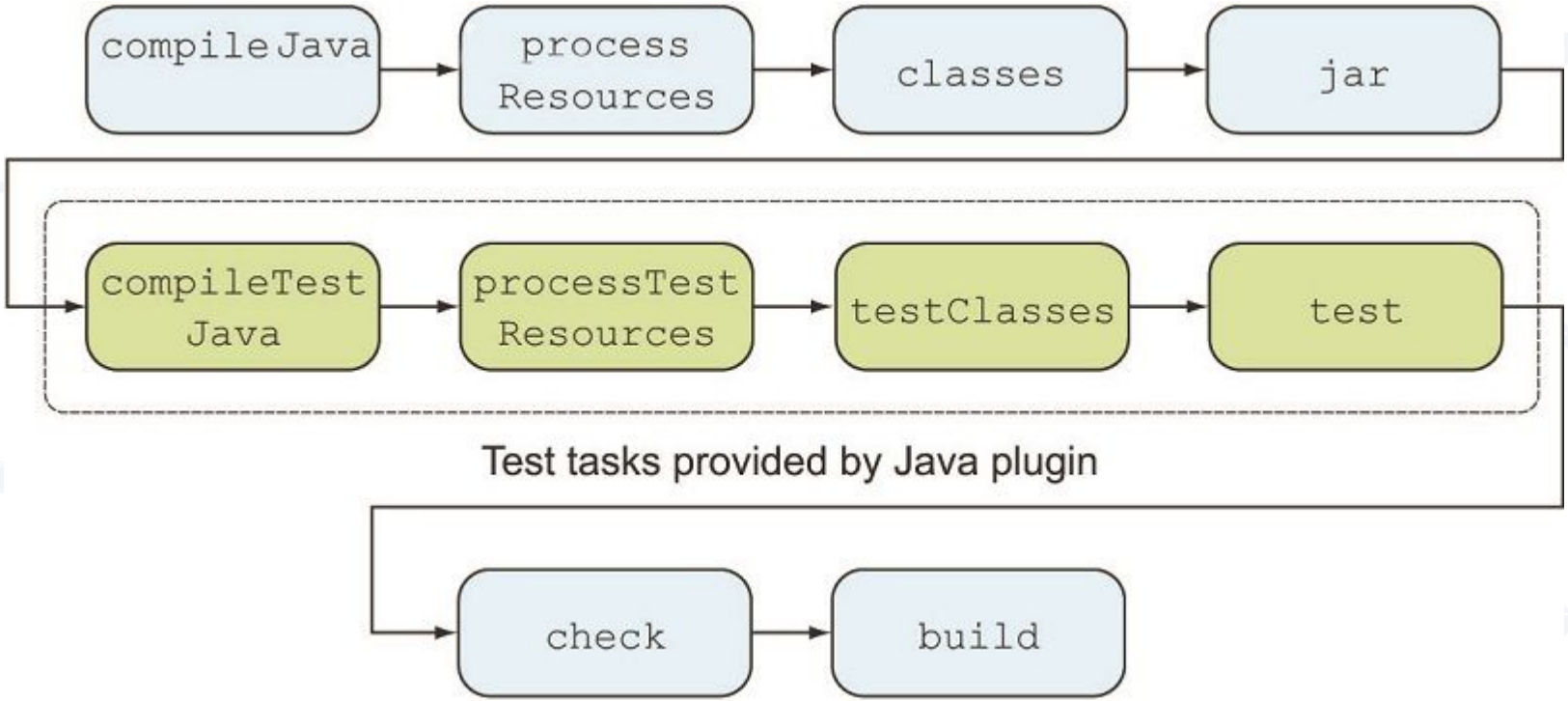
Wrapper plugin — an easy way to ensure a user of your Maven build has everything that is necessary.

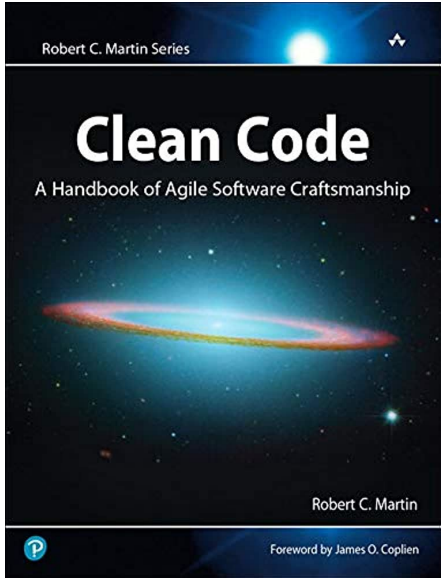
Spring Boot plugin — compiles your Spring Boot app, build an executable fat jar.

Exec — amazing general purpose plugin, can run arbitrary commands :)

The big picture




compileJava
process Resources
classes
jar
compileTest Java
processTest Resources
testClasses
test
Test tasks provided by Java plugin
check
build



Код є чистим, якщо його легко зрозуміти – усім членам команди. Чистий код може бути прочитаний і покращений розробником, відмінним від його оригінального автора. Зі зрозумілістю приходить читабельність, змінність, розширюваність і ремонтопридатність.

General rules

1. Follow standard conventions.
2. Keep it simple stupid. Simpler is always better. Reduce complexity as much as possible.
3. Boy scout rule. Leave the campground cleaner than you found it.
4. Always find root cause. Always look for the root cause of a problem.