รายงาน

เรื่อง การศึกษาสถาปัตยกรรมการออกแบบของ Jenkins

จัดทำโดย

62010381 ธนา	ติ้งประสม
62010533 ปรัชญา	ทองแสน
62010535 ปรัตถกร	ศรีบรรยงค์
62010634 พิม	ปิยจิรานันท์
62010694 ภากรณ์	ธนประชานนท์
62010978 สุมินชา	ชลอวงษ์

เสนอ

อาจารย์ ปริญญา เอกปริญญา

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา

01076024 Software Architecture and Design

ภาคการศึกษาที่ 1 ปีการศึกษา 2564

คณะวิศวกรรมศาสตร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

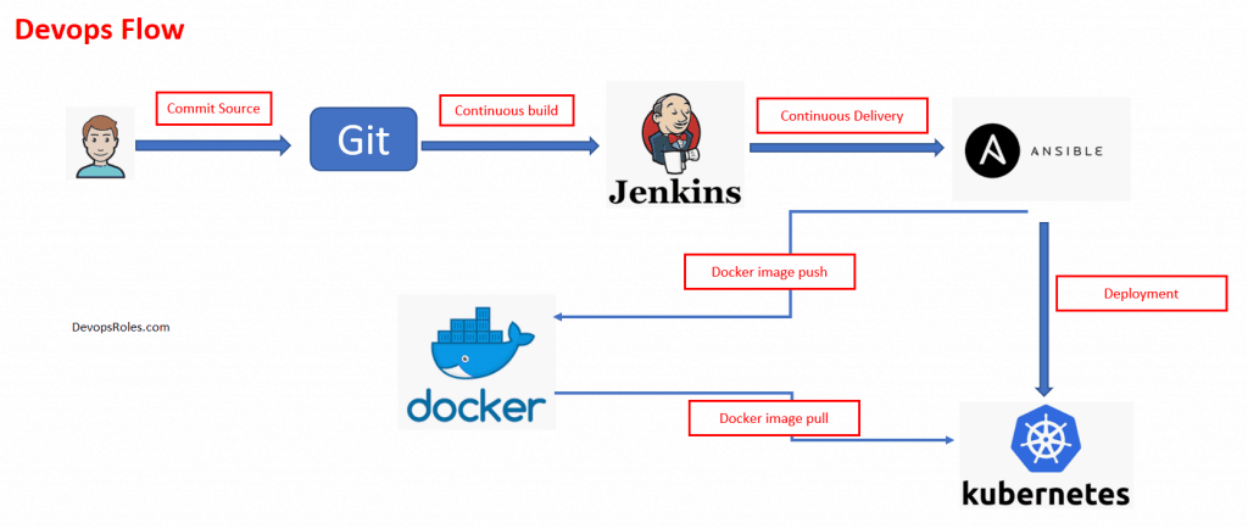
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

สารบัญ

Jenkins คืออะไร

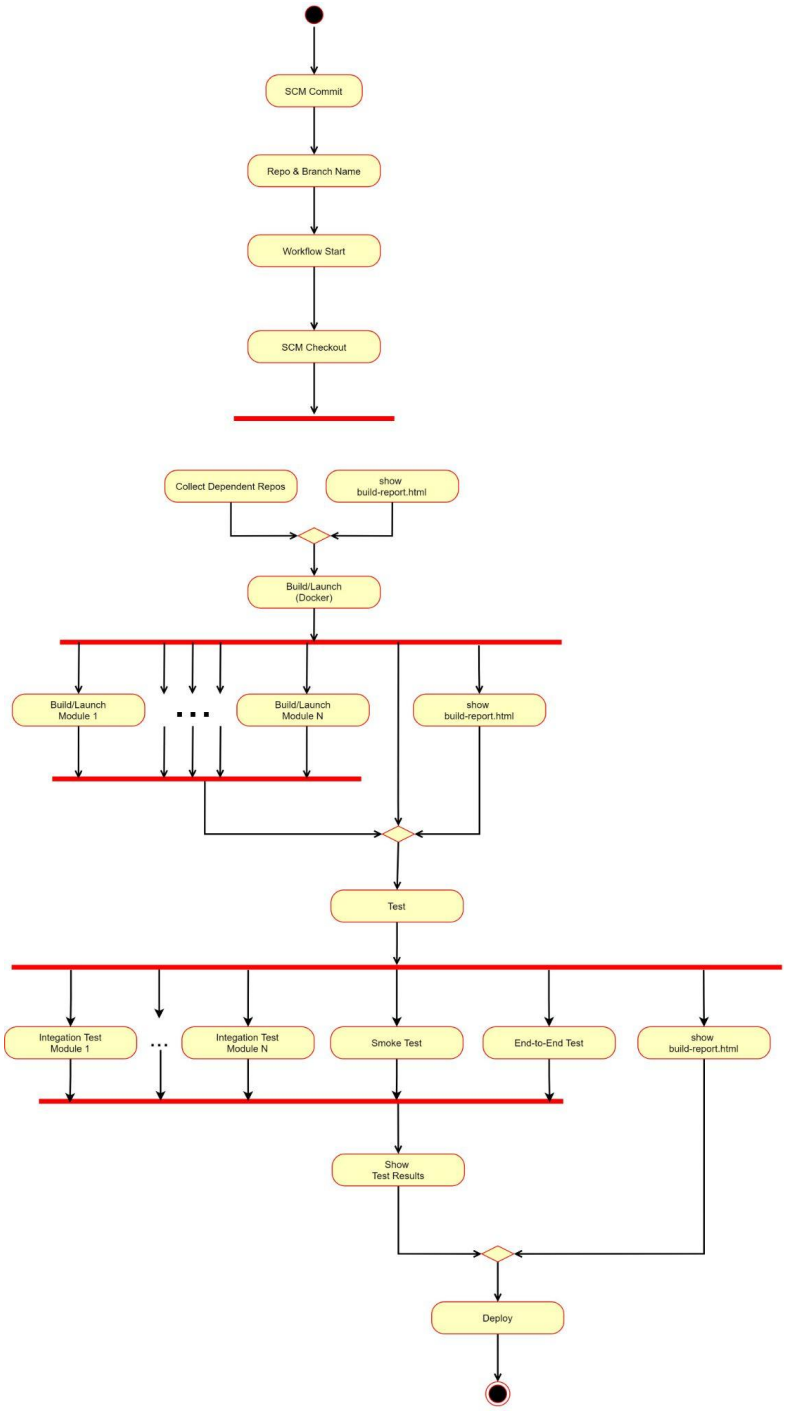
Jenkins คือ Software open-source ที่ถูกพัฒนาด้วยภาษา Java ใช้สำหรับทำ CI/CD (Continuous Integration/Continuous Delivery) ช่วยทำให้นักพัฒนาสามารถที่จะผลิตและส่งมอบ Software ไปยังผู้ใช้ได้อย่างต่อเนื่อง โดยการทำทุกอย่างให้เป็นไปอย่างอัตโนมัติ โดยไม่ต้องใช้มนุษย์มาสั่งการ Jenkins เหมาะสำหรับการใช้งานในโปรเจคขนาดเล็กไปจนถึงขนาดใหญ่ระดับองค์กร และยังสามารถนำไปใช้งานร่วมกับ open-source หรือ enterprise tools อื่น ๆ ได้อย่างง่ายดาย เช่น Github, Gitlab, Docker, AWS เป็นต้น

ตัวอย่างการใช้งาน Jenkins



Architecture Diagram

Jenkins ใช้ : Pipes and Filters architecture



Link: [Archtitecture.jpg (github)]

จุดอ่อนของ Architecture

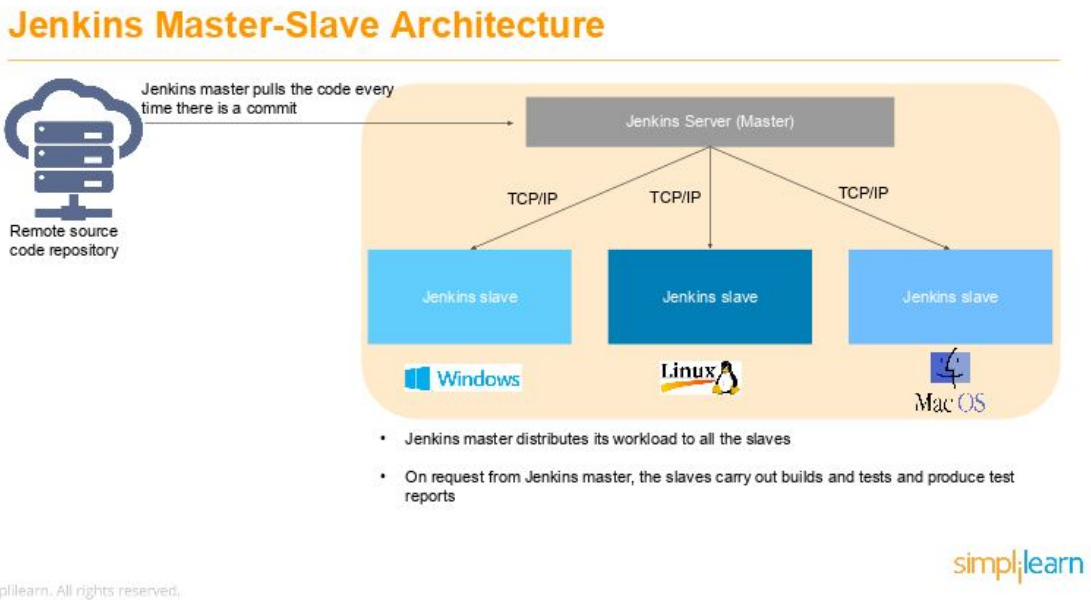
1. Interactive transformations ทำได้ยาก เนื่องจากตัว architecture เองไม่สนับสนุน interactive system อยู่แล้ว
2. มี Independent filters เยอะทำให้เกิด computation overhead เกิด Latency
3. ไม่สนับสนุน long-running computations

วิธีพัฒนาจุดอ่อนของ Architecture

เพิ่ม Throughput ของ Pipes and filter โดยใช้ Task farm parallelization pattern (เป็นการสร้างและจัดการ instances ของ filter แบบ parallel) อ่านเพิ่มเติม

Quality attributes

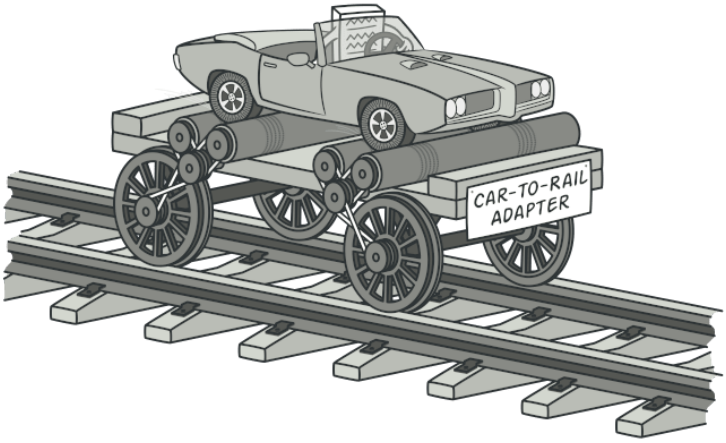
1. Scalability
    a. นักพัฒนาเลือกใช้ QA นี้เพราะว่า Jenkins นั้น Support Master-Slave
    b. ออกแบบให้มีการใช้งาน protocol TCP/IP ในการติดต่อสื่อสารระหว่างกันเองของ jenkins เพื่อให้สามารถทำตามคำสั่งเมื่อมีการสั่งงานมาจาก Master-Node
    c. [Jenkins Master and Slave Architecture – A Complete Guide]



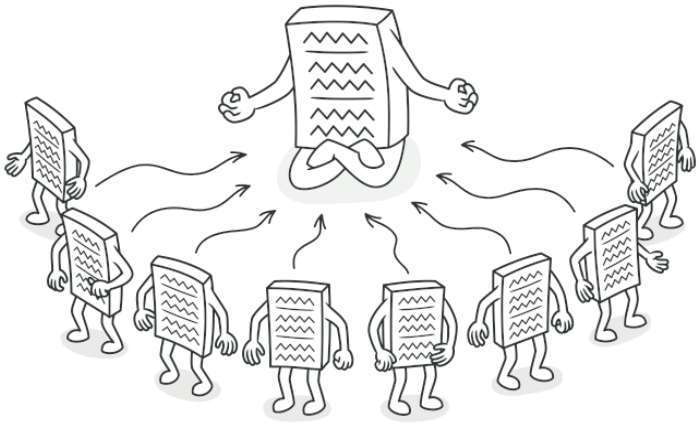
2. Extensibility
    a. นักพัฒนาเลือกใช้ QA นี้เพราะว่า Jenkins object model สามารถที่จะ extend ได้ (เช่นเราสามารถที่จะเพิ่ม SCM implementation จาก interface ที่ jenkins ให้มา)
    b. Jenkins จะให้ interface มาเพื่อให้นักพัฒนามาพัฒนาต่อได้
    c. [Writing an SCM Plugin]
3. Portability
    a. Jenkins ต้องสามารถใช้งานได้ในหลายๆ platform
    b. Jenkins เขียนด้วยภาษา JAVA ที่รันบน JVM ทำให้สามารถ cross platform ได้
    c. [Jenkins รันได้ทุกที่ที่มี JAVA Runtime Environment (JRE)]

Design pattern

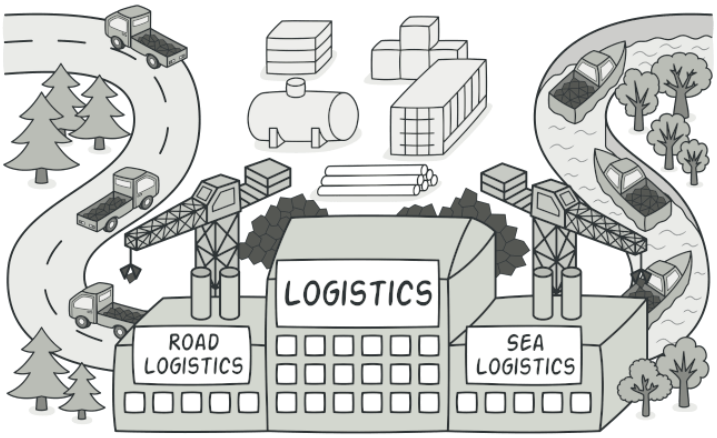
1. **Adapter** เป็น structural design pattern ที่ทำให้ objects ที่เข้ากันไม่ได้สามารถที่จะทำงานด้วยกันได้



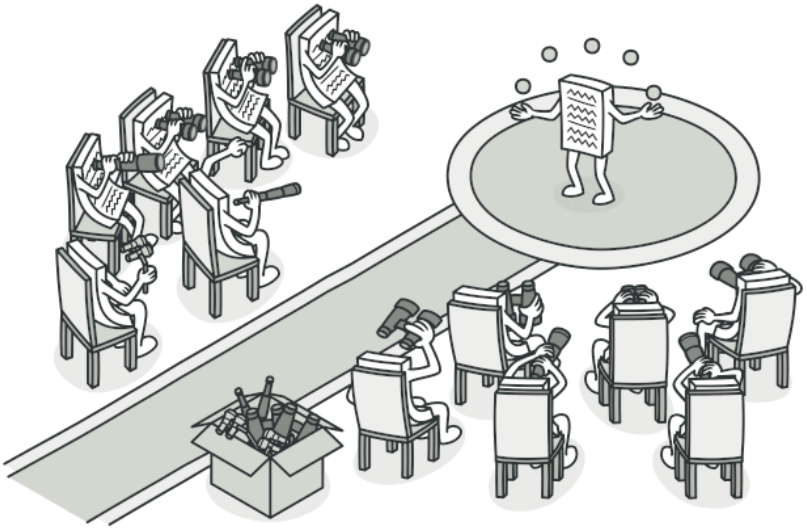
2. **Singleton** เป็น creational design pattern ที่ทำให้เวลาที่จะเข้าถึง class ต้องเข้าถึงผ่านจุดๆเดียวเท่านั้น



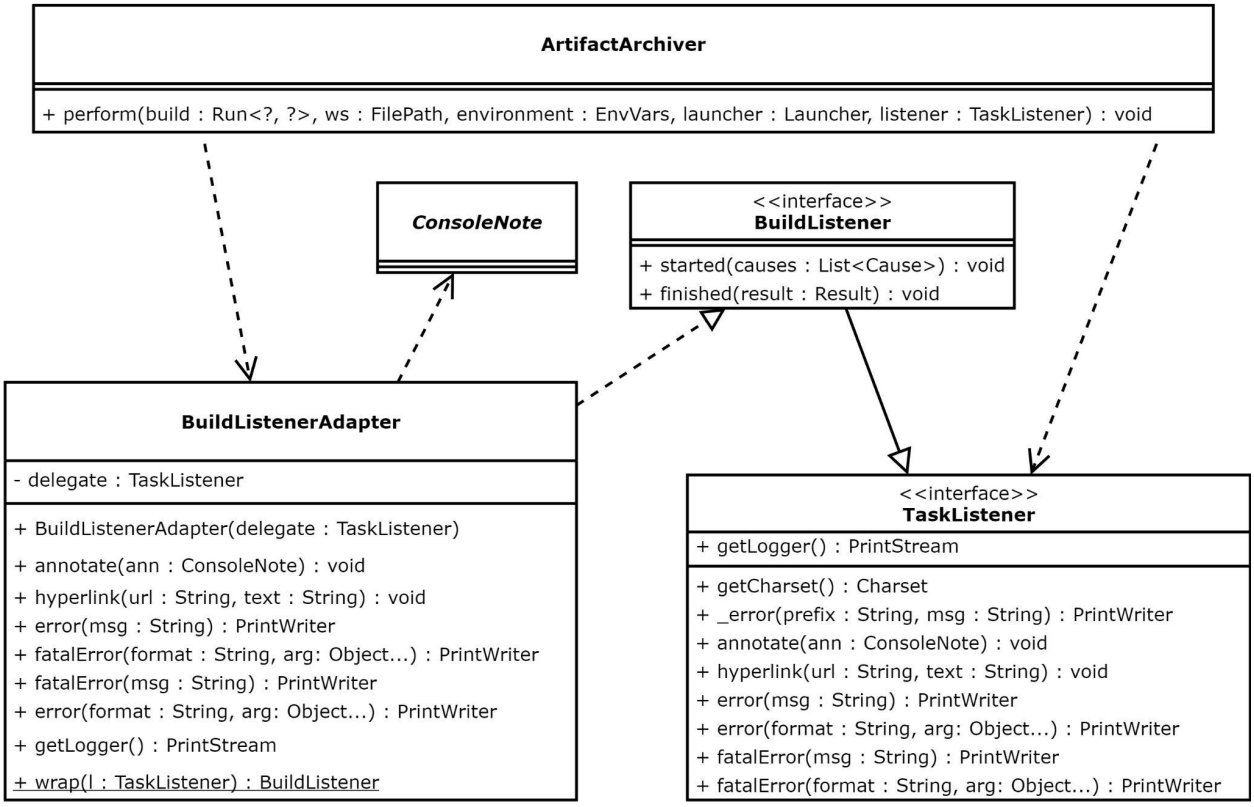
3. **Factory Method** เป็น creational design pattern ที่ให้ interface สำหรับ superclass แต่สามารถให้ subclass ปรับแต่งประเภทของ objects ที่จะสร้างได้



4. **Observer** เป็น Behavioral design pattern ที่อนุญาตให้กำหนด กลไก subscription เพื่อที่จะแจ้งเตือน objects หลายๆตัว เมื่อมี events เกิดขึ้นกับตัวที่ objects เหล่านั้น observe อยู่



Adapter



Link: Adapter.png (github)

```
public final class BuildListenerAdapter implements BuildListener {

    private final TaskListener delegate;

    public BuildListenerAdapter(TaskListener delegate) {
        this.delegate = delegate;
    }

    @Override public PrintStream getLogger() {
        return delegate.getLogger();
    }

    @SuppressWarnings("rawtypes")
    @Override public void annotate(ConsoleNote ann) throws IOException {
        delegate.annotate(ann);
    }

    @Override public void hyperlink(String url, String text) throws IOException {
        delegate.hyperlink(url, text);
    }

    @Override public PrintWriter error(String msg) {
        return delegate.error(msg);
    }

    @Override public PrintWriter error(String format, Object... args) {
        return delegate.error(format, args);
    }
```

```java
public interface BuildListener extends TaskListener {

    /**
     * Called when a build is started.
     *
     * @param causes
     *      Causes that started a build. See {@link Run#getCauses()}.
     */
    default void started(List<Cause> causes) {
        PrintStream l = getLogger();
        if (causes == null || causes.isEmpty()) {
            l.println("Started");
        } else {
            for (Cause cause : causes) {
                // TODO elide duplicates as per CauseAction.getCauseCounts (used in summary.
                jelly)
                cause.print(this);
            }
        }
    }

    /**
     * Called when a build is finished.
     */
    default void finished(Result result) {
        getLogger().println("Finished: " + result);
    }

}
```

```java
public interface TaskListener extends SerializableOnlyOverRemoting {
    /**
     * This writer will receive the output of the build
     */
    @NonNull
    PrintStream getLogger();

    /**
     * A charset to use for methods returning {@link PrintWriter}.
     * Should match that used to construct {@link #getLogger}.
     * @return by default, UTF-8
     */
    @Restricted(ProtectedExternally.class)
    @NonNull
    default Charset getCharset() {
        return StandardCharsets.UTF_8;
    }

    @Restricted(NoExternalUse.class) // TODO Java 9 make private
    default PrintWriter _error(String prefix, String msg) {
        PrintStream out = getLogger();
        out.print(prefix);
        out.println(msg);

        Charset charset = getCharset();
        return new PrintWriter(new OutputStreamWriter(out, charset), true);
    }
```

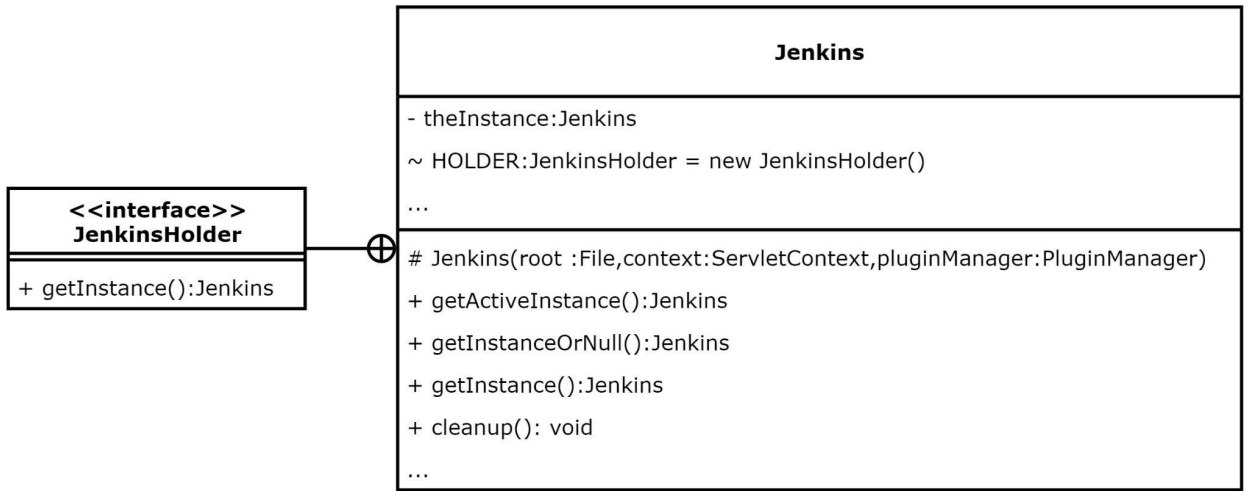
Link: BuilderListenerAdapter (github)

Link: BuildListener (github)

Link: TaskListener (github)

Singleton

Singleton



Link: Singleton.jpg (github)

```java
public static Jenkins get() throws IllegalStateException {
    Jenkins instance = getInstanceOrNull();
    if (instance == null) {
        throw new IllegalStateException("Jenkins.instance is missing. Read the
        documentation of Jenkins.getInstanceOrNull to see what you are doing
        wrong.");
    }
    return instance;
}
```

```java
@CheckForNull
public static Jenkins getInstanceOrNull() {
    return HOLDER.getInstance();
}
```

```
757     static JenkinsHolder HOLDER = new JenkinsHolder() {
758         public @CheckForNull Jenkins getInstance() {
759             return theInstance;
760         }
761     };
```

```
public static Jenkins getInstance() {
  return getInstanceOrNull();
}
```

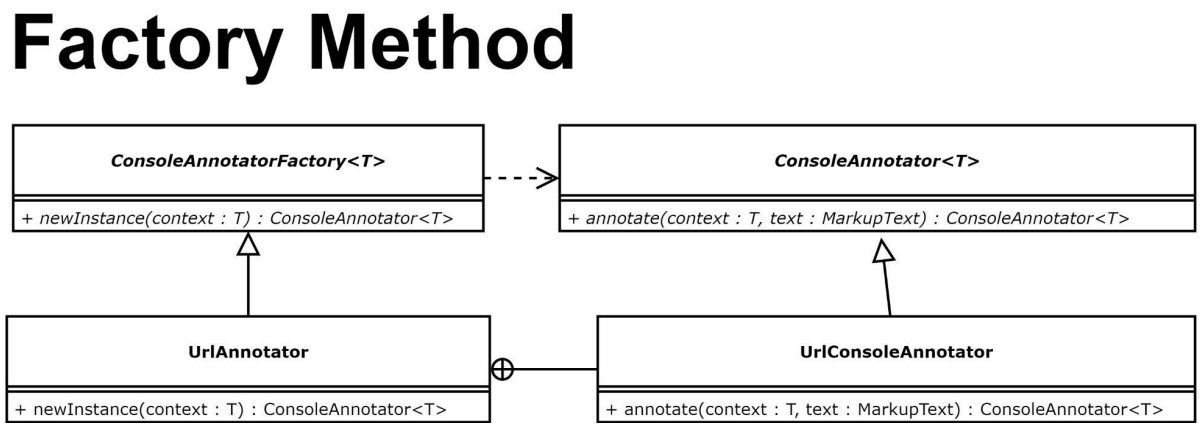
Link: Jenkins get (github)

Link: getInstanceOrNull (github)

Link: HOLDER (github)

Link: getInstance (github)

Factory Method



Link: Factorymethod.png (github)

```java
public abstract class ConsoleAnnotatorFactory<T> implements ExtensionPoint {
    /**
     * Called when a console output page is requested to create a stateful {@link ConsoleAnnotator}.
     *
     * <p>
     * This method can be invoked concurrently by multiple threads.
     *
     * @param context
     *      The model object that owns the console output, such as {@link Run}.
     *      This method is only called when the context object if assignable to
     *      {@linkplain #type() the advertised type}.
     * @return
     *      null if this factory is not going to participate in the annotation of this console.
     */
    public abstract ConsoleAnnotator<T> newInstance(T context);

    /**
     * For which context type does this annotator work?
     */
    public Class<?> type() {
        Type type = Types.getBaseClass(getClass(), ConsoleAnnotatorFactory.class);
        if (type instanceof ParameterizedType)
            return Types.erasure(Types.getTypeArgument(type,0));
        else
            return Object.class;
    }
```

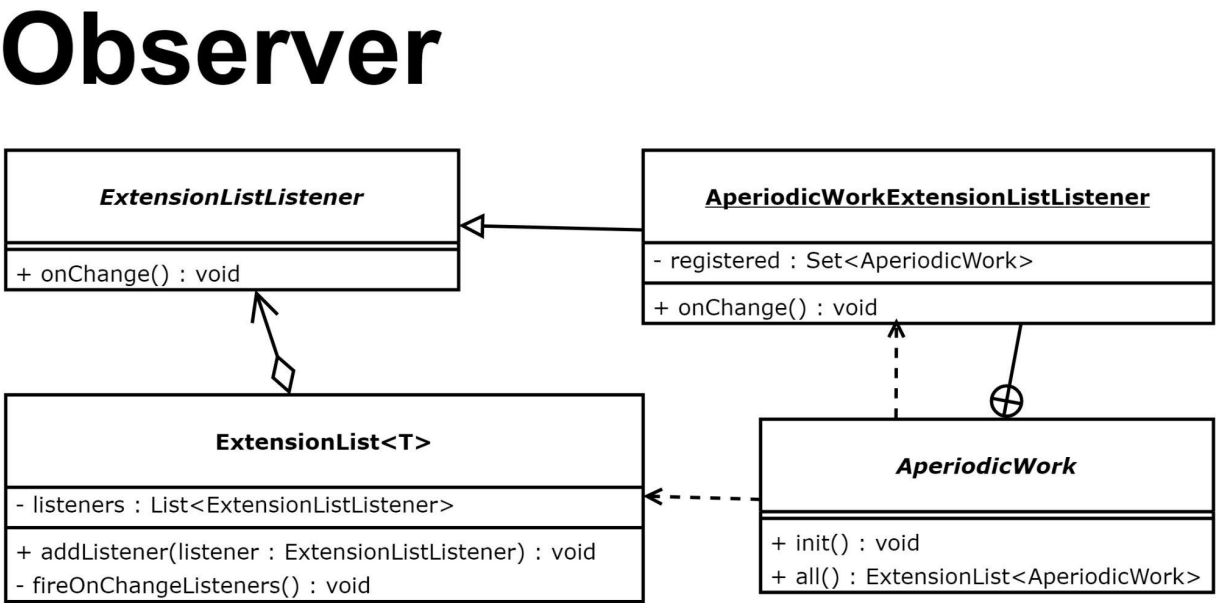
Link: ConsoleAnnotatorFactory (github)

Link: ConsoleAnnotator (github)

Link: UrlAnnotator (github)

Link: UrlConsoleAnnotator (github)

Observer



Link: Observer.png (github)

```java
public abstract class ExtensionListListener {

    /**
     * {@link ExtensionList} contents has changed.
     * <p>
     * This would be called when an entry gets added to or removed from the li
st for any reason e.g.
     * when a dynamically loaded plugin introduces a new {@link ExtensionPoin
t} implementation
     * that adds an entry to the {@link ExtensionList} being listened to.
     */
    public abstract void onChange();
}
```

```java
public abstract class AperiodicWork extends SafeTimerTask implements ExtensionPoint {

    @Initializer(after= JOB_CONFIG_ADAPTED)
    public static void init() {
        // start all AperidocWorks
        ExtensionList<AperiodicWork> extensionList = all();
        extensionList.addListener(new AperiodicWorkExtensionListListener(extensionList
));
        for (AperiodicWork p : AperiodicWork.all()) {
            scheduleAperiodWork(p);
        }
    }
    private static class AperiodicWorkExtensionListListener extends
ExtensionListListener {

        private final Set<AperiodicWork> registered = new HashSet<>();

        AperiodicWorkExtensionListListener(ExtensionList<AperiodicWork>
initiallyRegistered) {
            registered.addAll(initiallyRegistered);
        }

        @Override
        public void onChange() {
            synchronized (registered) {
                for (AperiodicWork p : AperiodicWork.all()) {
                    if (!registered.contains(p)) {
                        scheduleAperiodWork(p);
                        registered.add(p);
                    }
                }
            }
        }
    }
}
```

```java
public class ExtensionList<T> extends AbstractList<T> implements OnMaster {

    private final List<ExtensionListListener> listeners = new CopyOnWriteArrayList<>();

    public void addListener(@NonNull ExtensionListListener listener) {
        listeners.add(listener);
    }

    private void fireOnChangeListeners() {
        for (ExtensionListListener listener : listeners) {
            try {
                listener.onChange();
            } catch (Exception e) {
                LOGGER.log(Level.SEVERE, "
Error firing ExtensionListListener.onChange().", e);
            }
        }
    }
}
```

Link: ExtensionListListener.java (github)
Link: AperiodicWork.java (github)
Link: ExtensionList.java (github)

อ้างอิง

- https://www.jenkins.io/pipeline/getting-started-pipelines/
- https://syedhasan010.medium.com/pipe-and-filter-architecture-bd7babdb908
- https://refactoring.guru/design-patterns
- https://www.simplilearn.com/tutorials/jenkins-tutorial/what-is-jenkins
- https://wiki.jenkins-ci.org/display/JENKINS/Architecture.html
- https://ieeexplore.ieee.org/document/7497426
- https://www.researchgate.net/publication/304457001_Increasing_the_Throughput_of_Pipe-and-Filter_Architectures_by_Integrating_the_Task_Farm_Parallelization_Pattern
- https://www.interaction-design.org/literature/topics/usability