UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware
Knowledge Software

Technique de tests

Master ILSEN

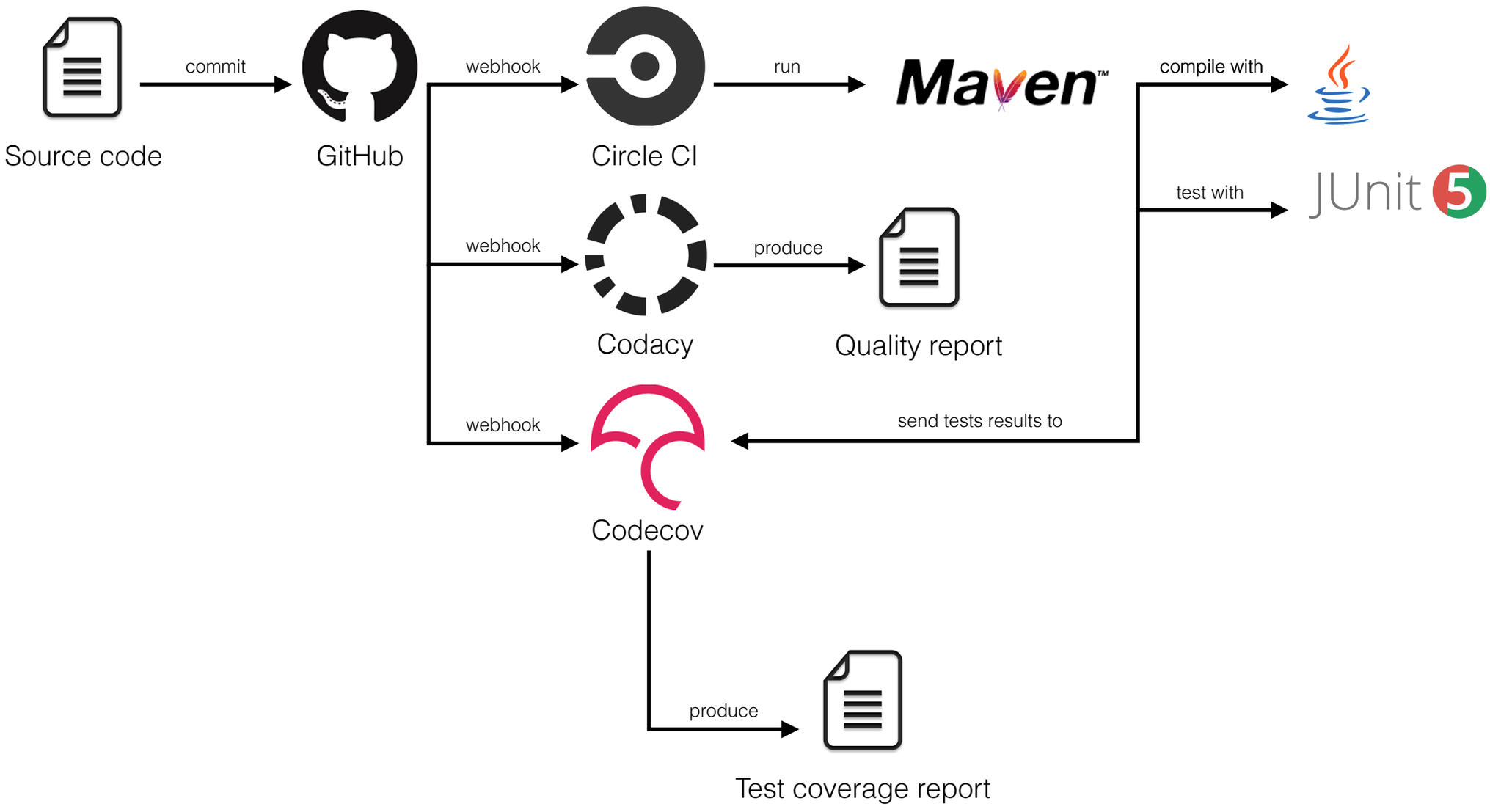
Introduction

Contact : felix.voituret@ismart.fr

- 6 heures de cours
- 12 heures de travaux pratiques

Evaluation : rendu de projet

Introduction



UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

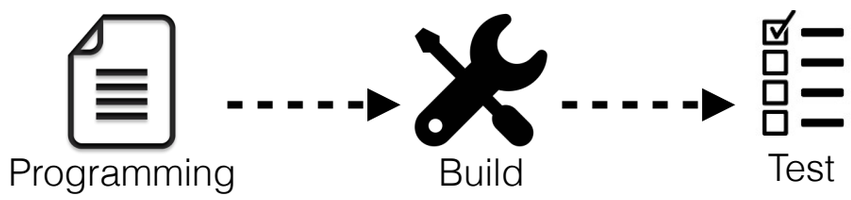
Kware
Knowledge Software

Intégration Continue

Maven / CircleCI

Intégration continue

L'**intégration continue** est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de **régression** dans l'application développée.



Apache Maven est un outil pour la gestion et l'automatisation de production logiciel. Les cycles de vie basique couvert sont les suivants :

- compile
- test
- package
- install
- deploy

```
> mvn goal
```

Un projet **Maven** est défini par un fichier nommé *POM* (Project Object Model) qui défini la configuration des différents cycles de vie de l'application.

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>authority</groupId>
  <artifactId>project_name</artifactId>
  <version>1.0.0</version>
</project>
```

Project Object Model

Un projet **Maven** est défini par un fichier nommé *POM* (Project Object Model) qui défini la configuration des différents cycles de vie de l'application.

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>authority</groupId>
  <artifactId>project_name</artifactId>
  <version>1.0.0</version>
</project>
```

Project Object Model

Un projet **Maven** est défini par un fichier nommé *POM* (Project Object Model) qui défini la configuration des différents cycles de vie de l'application.

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>authority</groupId>
  <artifactId>project_name</artifactId>
  <version>1.0.0</version>
</project>
```

Project Object Model

Un projet **Maven** est défini par un fichier nommé *POM* (Project Object Model) qui défini la configuration des différents cycles de vie de l'application.

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>authority</groupId>
  <artifactId>project_name</artifactId>
  <version>1.0.0</version>
</project>
```

Project Object Model

Un projet **Maven** est défini par un fichier nommé *POM* (Project Object Model) qui défini la configuration des différents cycles de vie de l'application.

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>authority</groupId>
  <artifactId>project_name</artifactId>
  <version>1.0.0</version>
</project>
```

Les paramètres de constructions sont indiqués dans un noeud build. Des profiles permettent de définir des configurations de build.

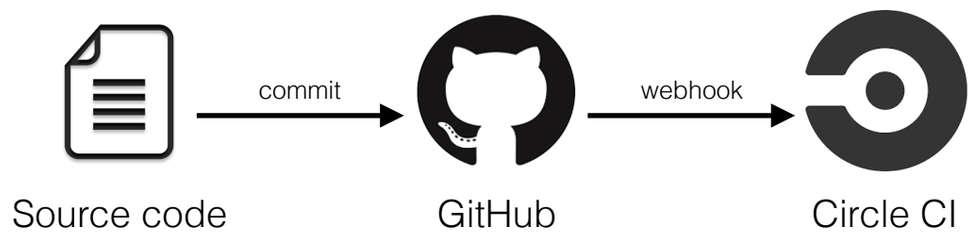
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  ...
  <build>...</build>
  <profiles>
    <profile>
      <build>...</build>
    </profile>
  </profiles>
  ...
</project>
```

Un grand nombre d'artifact maven sont disponible sur un repository nommée **Maven Central** et peuvent être intégré dans n'importe quel projet en spécifiant un noeud dependencies dans le *POM*.

```
<project xmlns="http://maven.apache.org/POM/4.0.0">
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
    </dependency>
  ...
  </dependencies>
  ...
</project>
```

CircleCI

CircleCI est une plateforme cloud offrant des services d'intégration continue dans des conteneurs. Un compte gratuit permet d'avoir un conteneur d'execution gratuitement relié à *n* repository (**Github** ou **Bitbucket**)



circle.yml

Après un commit sur votre repository **Github**. Un conteneur est crée, et cherche un fichier `.circleci/config.yml`, décrivant les étapes et la configuration de votre intégration continue :

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware
Knowledge Software

Test unitaire

JUnit

Test unitaire

Un **test unitaire** est un test s'assurant du bon fonctionnement d'une partie de code ou unité.

```java
public final class SumTest {

  @Test
  public void testSum() {
    final int sum = 2 + 2;
    assertEquals(4, sum);
  }

}
```

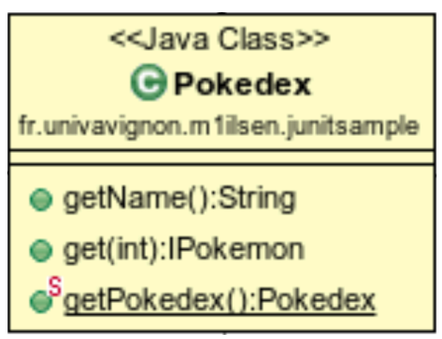
powered by JUnit 5

JUnit est un framework pour l'écriture et l'execution de test unitaire pour **Java,** intégré nativement dans la plupart des outils de développement actuel.

Test case : Test couvrant une portion de l'application.

Test suite : Ensemble de test case.

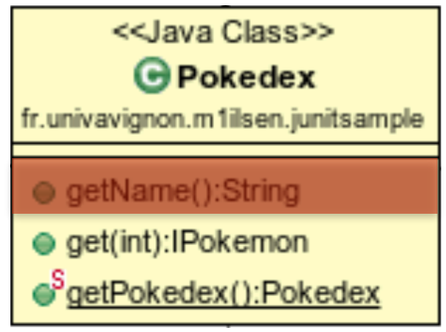
Test case


<<Java Class>>
Pokedex
fr.univavignon.m1ilsen.junitsample
getName():String
get(int):IPokemon
getPokedex():Pokedex

Test case

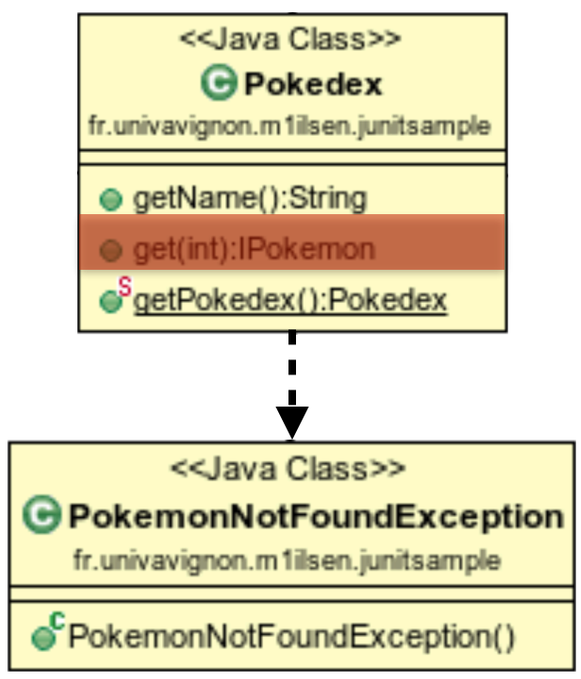
Un **test case** est une simple classe Java disposant d'au moins une méthode annoté avec @org.junit.Test :

```java
public final class PokedexTest {

  @Test
  public void testName() {
    final Pokedex pokedex = Pokedex.getPokedex();
    final String name = pokedex.getName();
    assertEquals("Dexter", name);
  }

}
```



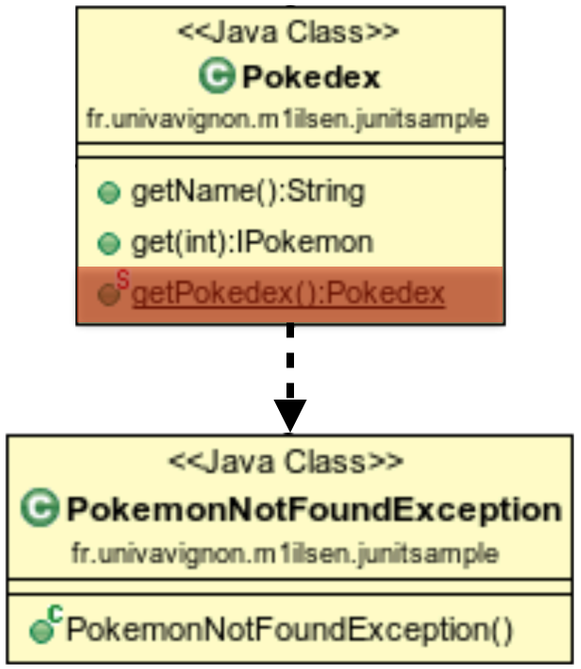
Pour tester les exceptions, un paramètre **expected** peut être donné à l'annotation @org.junit.Test :

```
public final class PokedexTest {

  @Test(expected=PokemonNotFoundException.class)
  public void testPokemonNotFoundException() {
    final Pokedex pokedex = Pokedex.getInstance();
    pokedex.get(-1);
  }

}
```



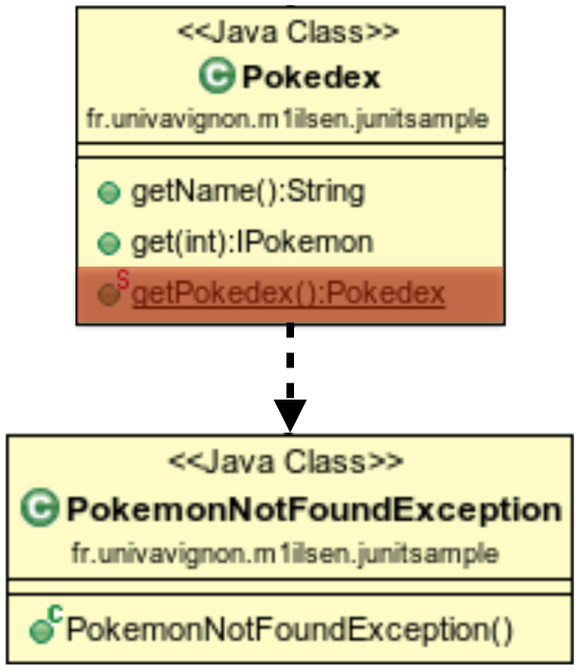
L'annotation @org.junit.Before permet de définir une méthode exécuté avant chaque test :

```java
public final class PokedexTest {

  private Pokedex pokedex;

  @Before
  public void setUp() {
     this.pokedex = Pokedex.getPokedex();
  }

  @Test
  public void testName() {
    assertEquals("Dexter", pokedex.getName());
  }

  @Test(expected=PokemonNotFoundException.class)
  public void testPokemonNotFoundException() {
    pokedex.get(-1);
  }

}
```



Un équivalent statique est l'annotation @org.junit.BeforeClass qui sera exécuté une seule fois avant tous les tests :

```java
public final class PokedexTest {

  private static Pokedex pokedex;

  @BeforeClass
  public static void setUp() {
     pokedex = Pokedex.getPokedex();
  }

  @Test
  public void testName() {
    assertEquals("Dexter", pokedex.getName());
  }

  @Test(expected=PokemonNotFoundException.class)
  public void testPokemonNotFoundException() {
    pokedex.get(-1);
  }

}
```

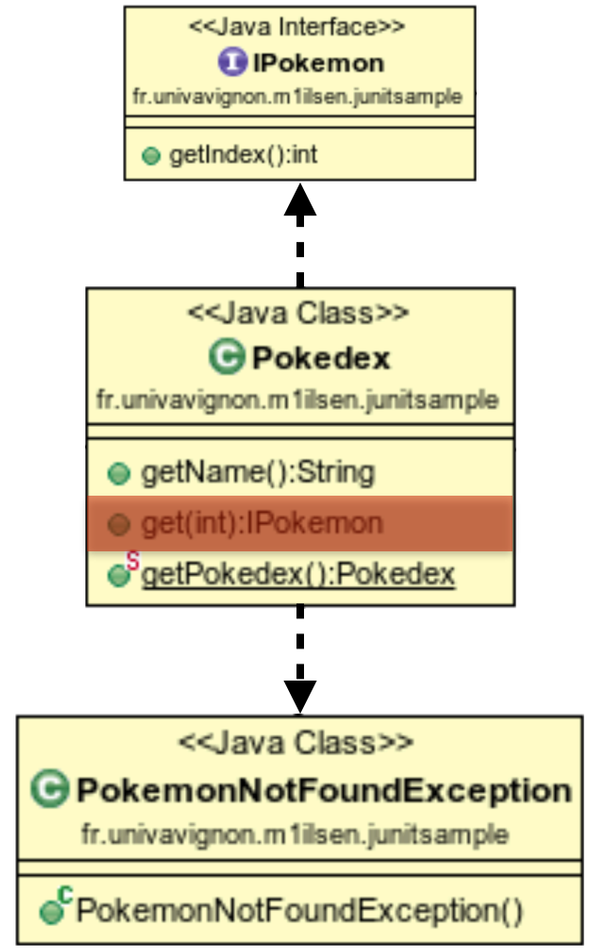


Un **mock** est une implémentation factice d'une interface permettant de tester ces dernières ainsi que leur interaction.


```java
public final class PokemonMock {

  @Override
  public int getIndex() {
    return 1;
  }

}
```

Test case

```java
public final class PokedexTest {

  private static Pokedex pokedex;

  @BeforeClass
  public static void setUp() {
    pokedex = Pokedex.getPokedex();
  }

  @Test
  public void testName() {
    assertEquals("Dexter", pokedex.getName());
  }

  @Test
  public void testGet() {
     final IPokemon pokemon = pokedex.get(1);
    assertEquals(1, pokemon.getIndex());
  }

  @Test(expected=PokemonNotFoundException.class)
  public void testPokemonNotFoundException() {
    pokedex.get(-1);
  }

}
```



Test case

Annotation disponible pour l'écriture d'un **test case** :

- @org.junit.Test
- @org.junit.Before
- @org.junit.After
- @org.junit.BeforeClass
- @org.junit.AfterClass

Test case

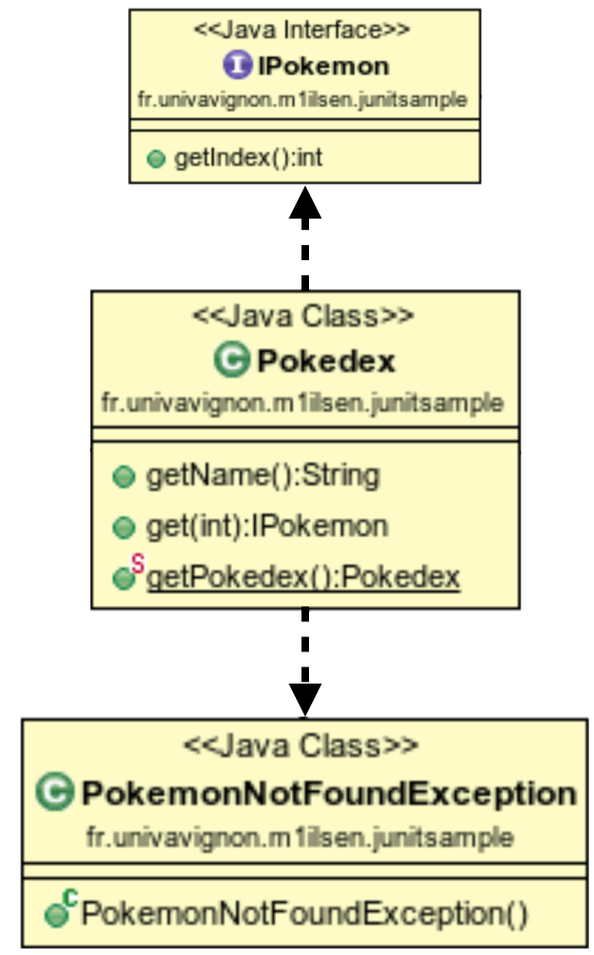
Méthodes disponible pour l'écriture d'un **test** :

- assertEquals
- assertTrue
- assertFalse
- assertNull
- assertNotNull
- fail

Test suite

Une **test suite** consiste en l'exécution de plusieurs **test cases** défini par l'annotation @org.junit.runner.Suite:

```java
@RunWith(Suite.class)
@Suite.SuiteClasses(
   PokedexTest.class,
   PokemonTest.class
)
public final class AllTests {

}
```



UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware
Knowledge Software

Mock

Mockito

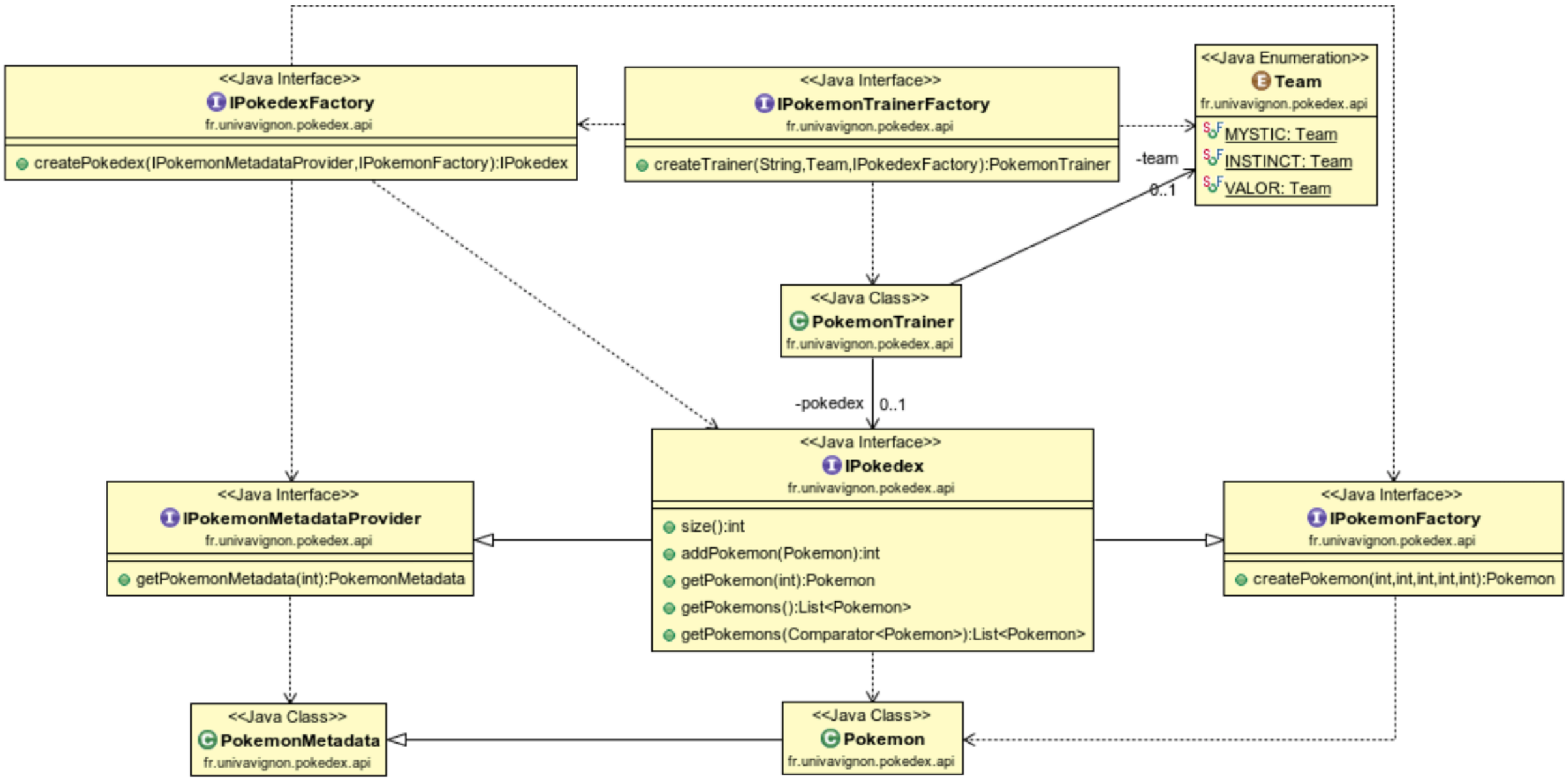
Un **mock** est une implémentation factice d'une interface permettant de tester ces dernières ainsi que leur interaction.


```java
public final class PokemonMock {

  @Override
  public int getIndex() {
    return 1;
  }

}
```

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware Knowledge Software

Introduction



Mockito

Mockito est un framework Java permettant de générer automatiquement des **mock**.
Pour l'intégrer à votre projet il suffit de rajouter la dépendance **Maven** suivante :

```xml
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-all</artifactId>
  <version>1.9.5</version>
</dependency>
```

Création de mock

Pour créer un **mock** il suffit d'annoter un attribut de la classe de test avec l'annotation @Mock.
La règle de test **MockitoRule** permet l'instantiation automatique des attributs annotés.

```
public final class PokemonTest {

  @Mock private IPokemon pokemonMock;

  @Rule public MockitoRule mockitoRule = MockitoJUnit.rule();

}
```

Configuration de mock

Mockito fourni des méthodes statiques pour configurer les résultats lors des appels de méthodes mockées.

```
when(pokemonMock.getName()).thenReturn(« Bulbizarre »)
```

```
when(pokemonMock.getName()).thenThrow(new Exception())
```

Verification de mock

Une fois les mocks utilisés dans vos tests unitaires, leur utilisation peut être contrôlée grace à la méthode statique verify.

```
verify(test).testing(Matchers.eq(12));
verify(test, times(2)).getUniqueId();
verify(mock, never()).someMethod("never called");
verify(mock, atLeastOnce()).someMethod("called at least once");
verify(mock, atLeast(2)).someMethod("called at least twice");
verify(mock, times(5)).someMethod("called five times");
verify(mock, atMost(3)).someMethod("called at most 3 times");
```

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware
Knowledge Software

Qualité de code

Codacy

Qualité de code

La qualité d’un logiciel est directement lié à la qualité du code

- Complexité
- Documentation
- Portabilité
- Architecture
- Code

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware
Knowledge Software

Object design

SOLID

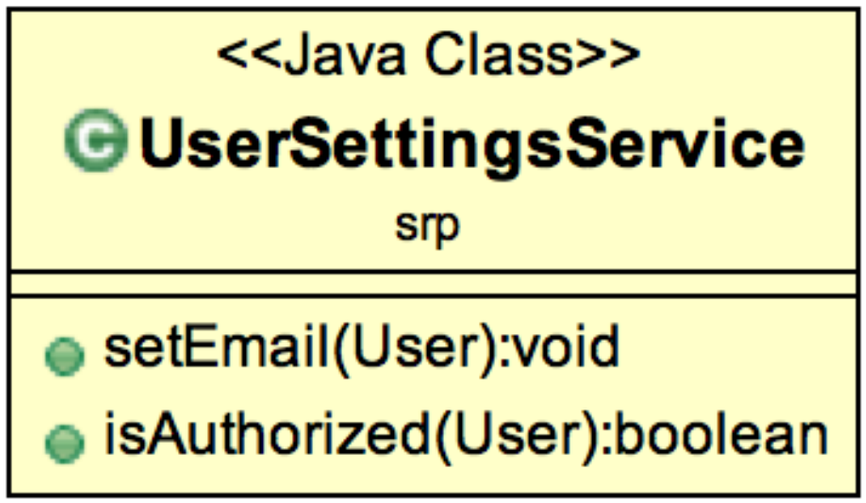
SOLID

SOLID est un acronyme dénotant les principes de base à appliquer pour une qualité de design objet satisfaisante :

- **S**ingle responsibility
- **O**pen / Closed
- **L**iskov substitution
- **I**nterface segregation
- **D**ependency inversion

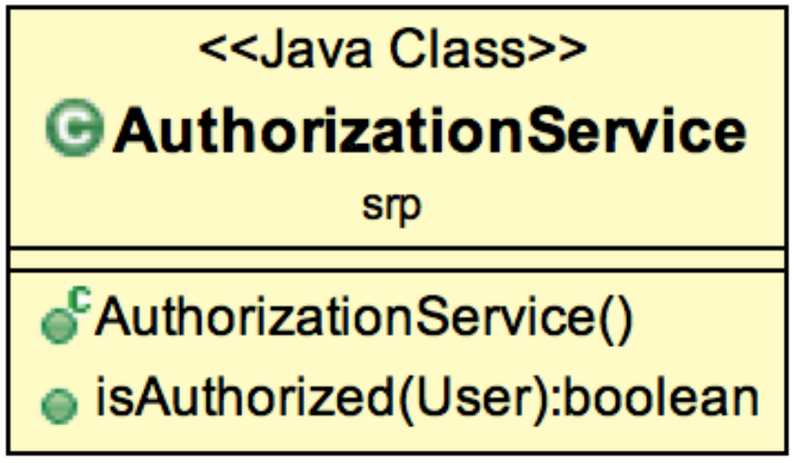
Single Responsibility Principle *(SRP)*

Il ne doit jamais avoir plus d'une seule raison de modifier une classe


<<Java Class>>
UserSettingsService
srp
setEmail(User):void
isAuthorized(User):boolean

SOLID

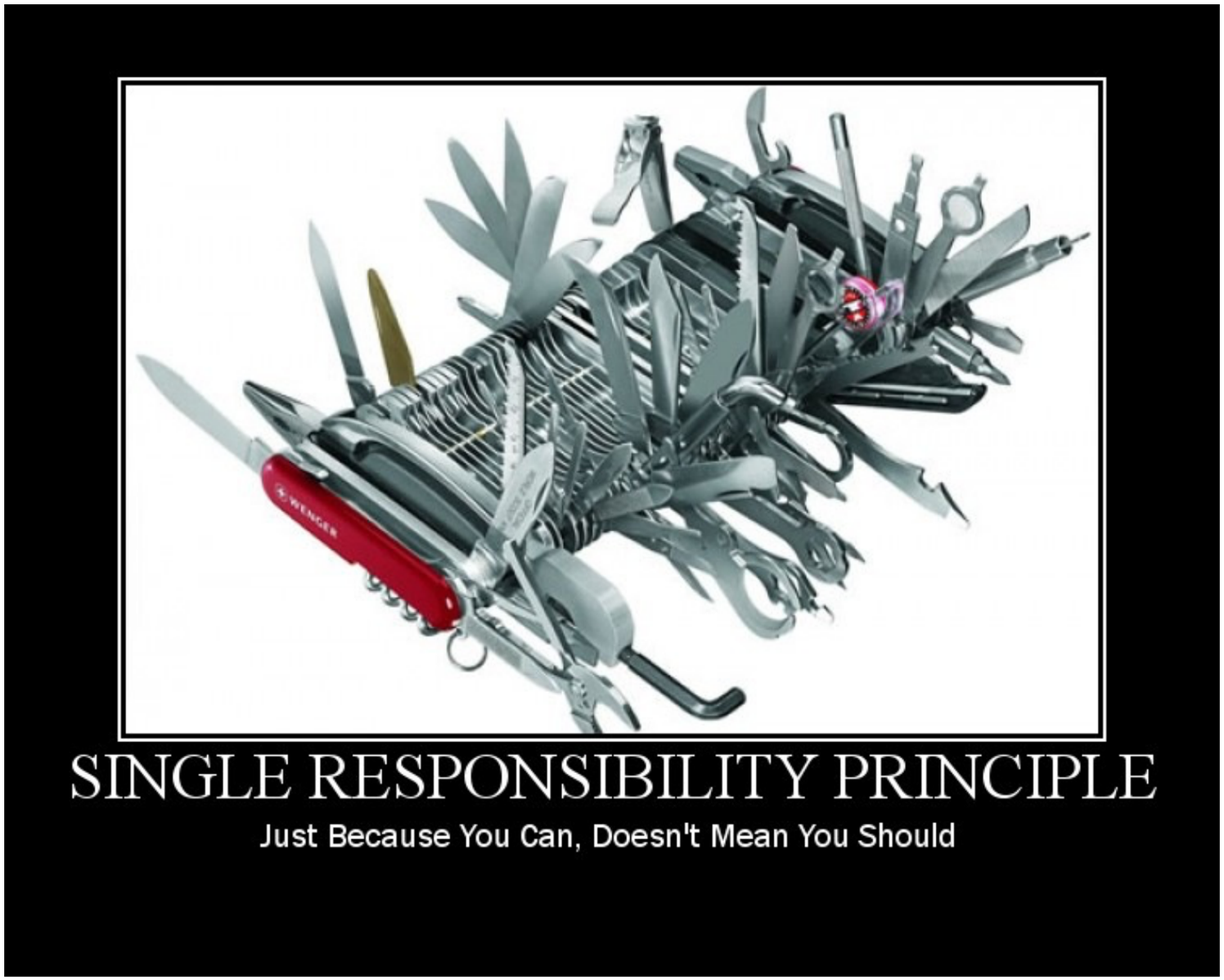

<<Java Class>>
UserSettingsService
srp
setEmail(User):void


<<Java Class>>
AuthorizationService
srp
AuthorizationService()
isAuthorized(User):boolean

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

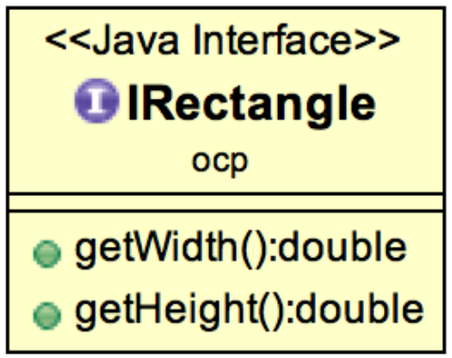
Kware
Knowledge Software

SOLID

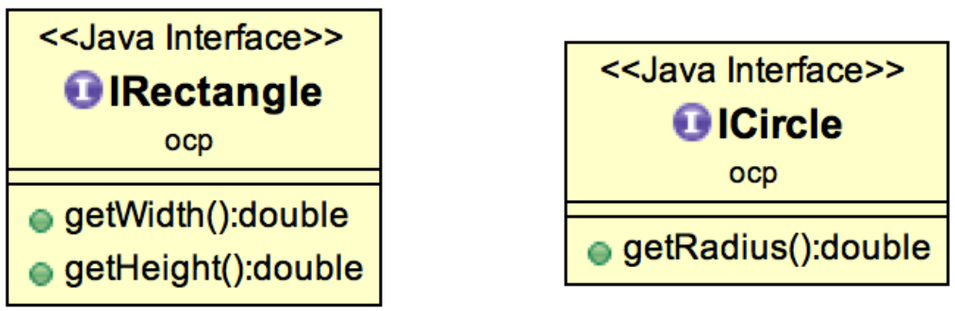


Open/Closed Principle *(OCP)*

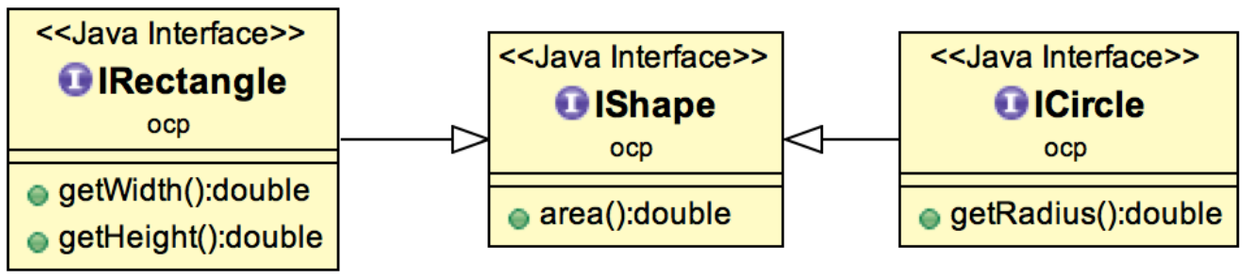
Une classe doit être ouverte à l'extension, et fermée à la modification




```java
public class AreaCalculator {

    public double area(Collection<IRectangle> rectangles) {
        double area = 0;
        for (final IRectangle rectangle : rectangles) {
            area += (rectangle.getWidth() * rectangle.getHeight());
        }
        return area;
    }

}
```




```java
public class AreaCalculator {

    public double area(Collection<Object> shapes) {
        double area = 0;
        for (final Object shape : shapes) {
            if (shape instanceof IRectangle) {
                final IRectangle rectangle = (IRectangle) shape;
                area += (rectangle.getWidth() * rectangle.getHeight());
            }
            else if (shape instanceof ICircle) {
                final ICircle circle = (ICircle) shape;
                area += (circle.getRadius() * circle.getRadius() * Math.PI);
            }
        }
        return area;
    }

}
```




```java
public class AreaCalculator {

    public double area(Collection<IShape> shapes) {
        double area = 0;
        for (final IShape shape : shapes) {
            area += shape.area();
        }
        return area;
    }

}
```

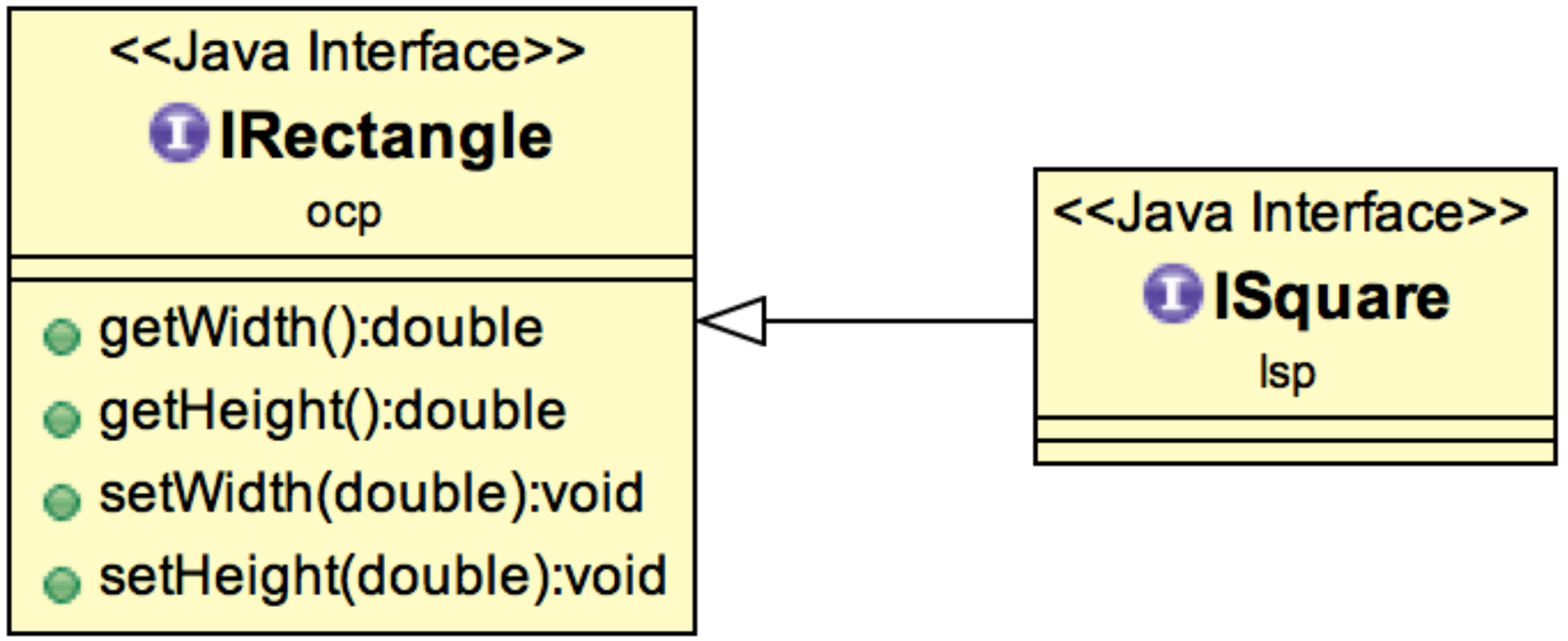
UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware
Knowledge Software

SOLID

Liskov Substitution Principle *(LSP)*

Une méthode qui utilise une référence de classe doit pouvoir utiliser des instances de classe dérivée sans les connaître

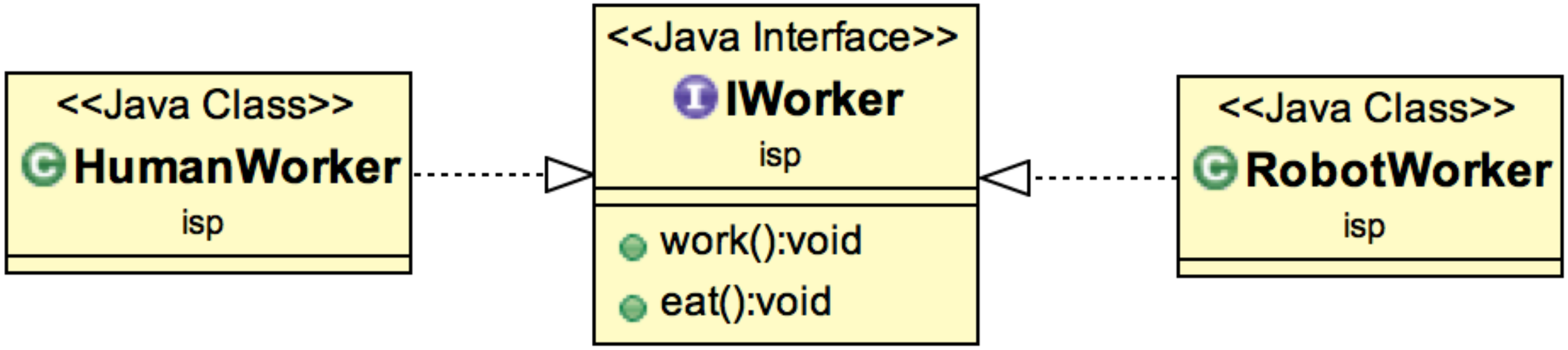

<<Java Interface>>
IRectangle
ocp
getWidth():double
getHeight():double
setWidth(double):void
setHeight(double):void
<<Java Interface>>
ISquare
lsp


LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

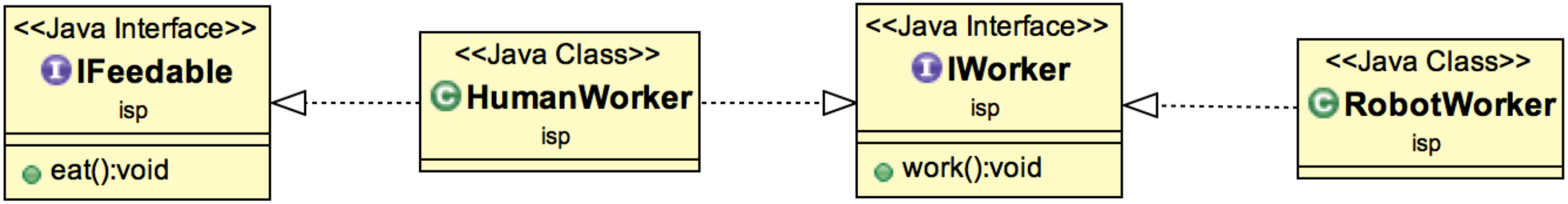
Interface Segregation Principle *(ISP)*

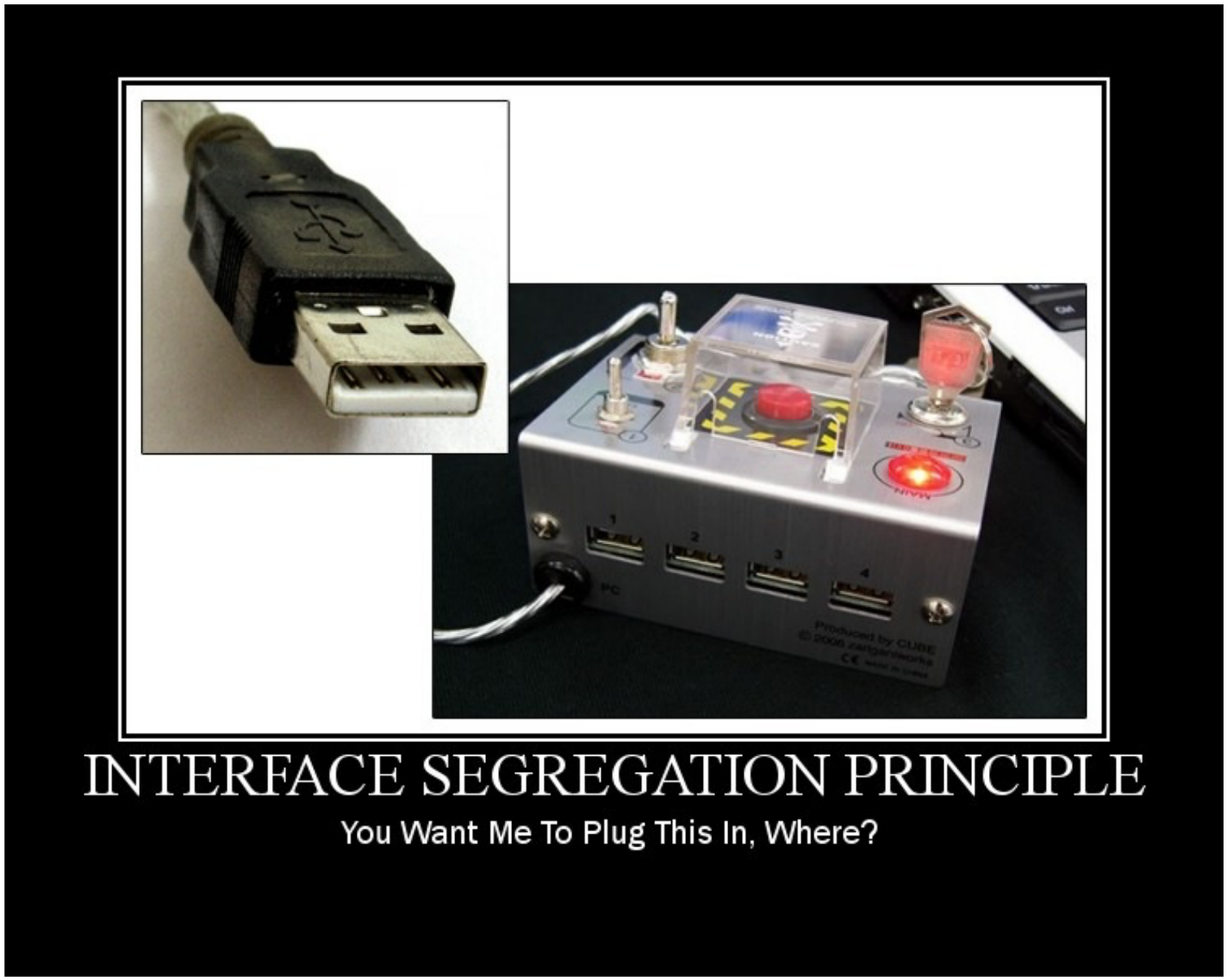
Plusieurs interfaces spécifiques sont meilleures qu'une seule interface générique

SOLID


<<Java Class>>
HumanWorker
isp
<<Java Interface>>
IWorker
isp
work():void
eat():void
<<Java Class>>
RobotWorker
isp

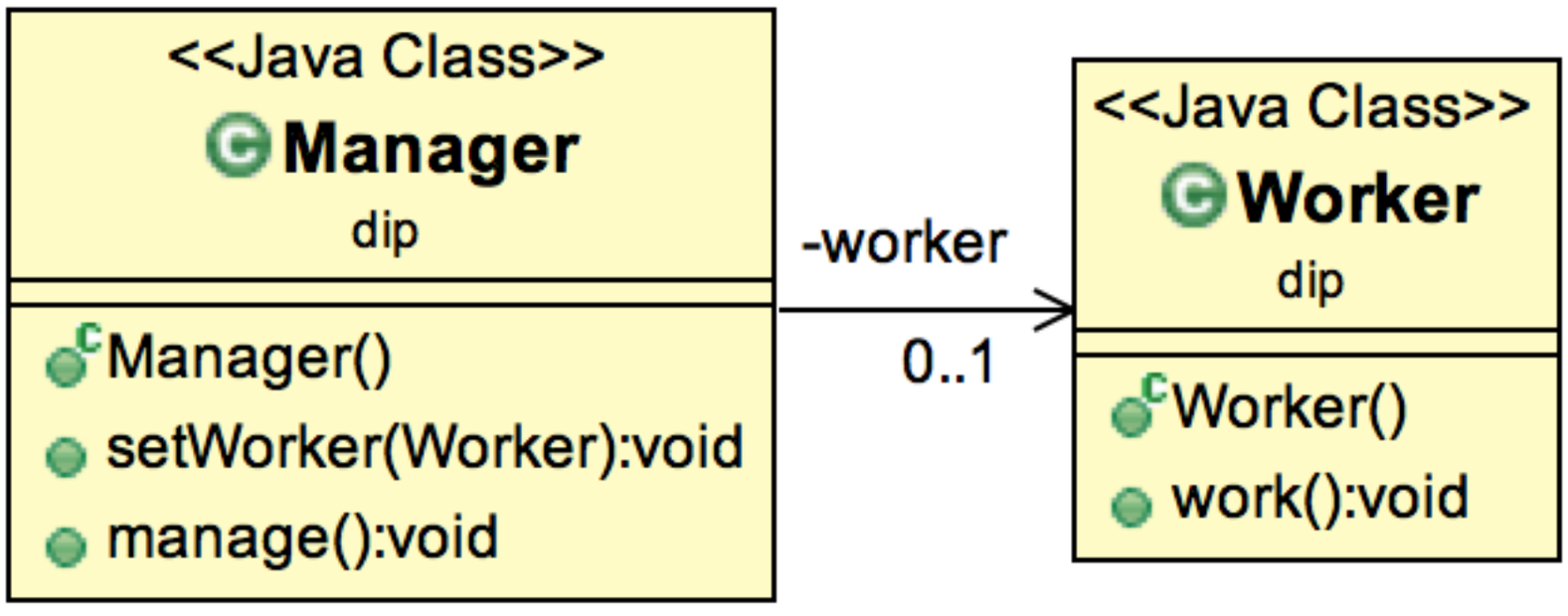
SOLID

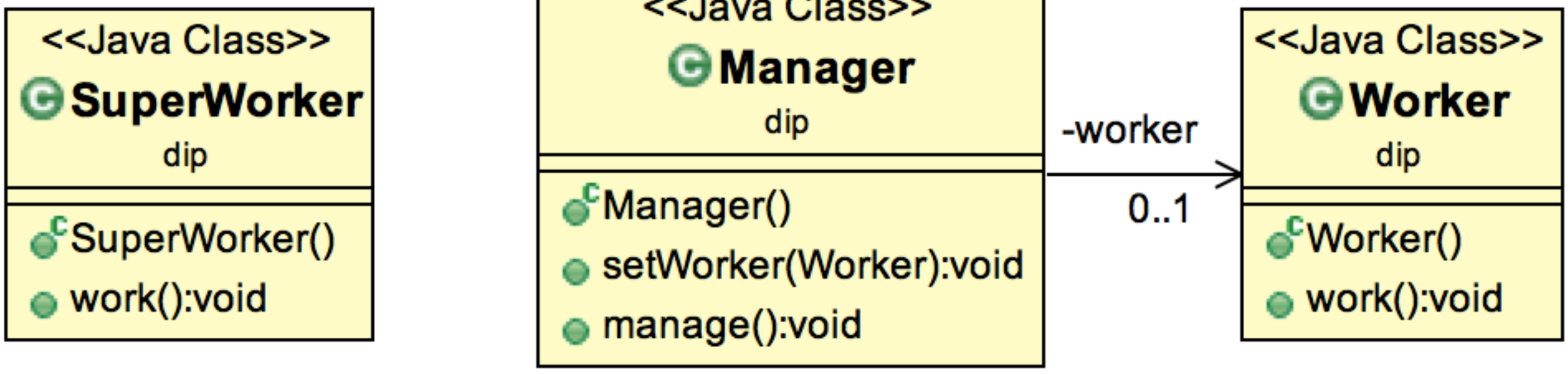

<<Java Interface>>
IFeedable
isp
eat():void
<<Java Class>>
HumanWorker
isp
<<Java Interface>>
IWorker
isp
work():void
<<Java Class>>
RobotWorker
isp

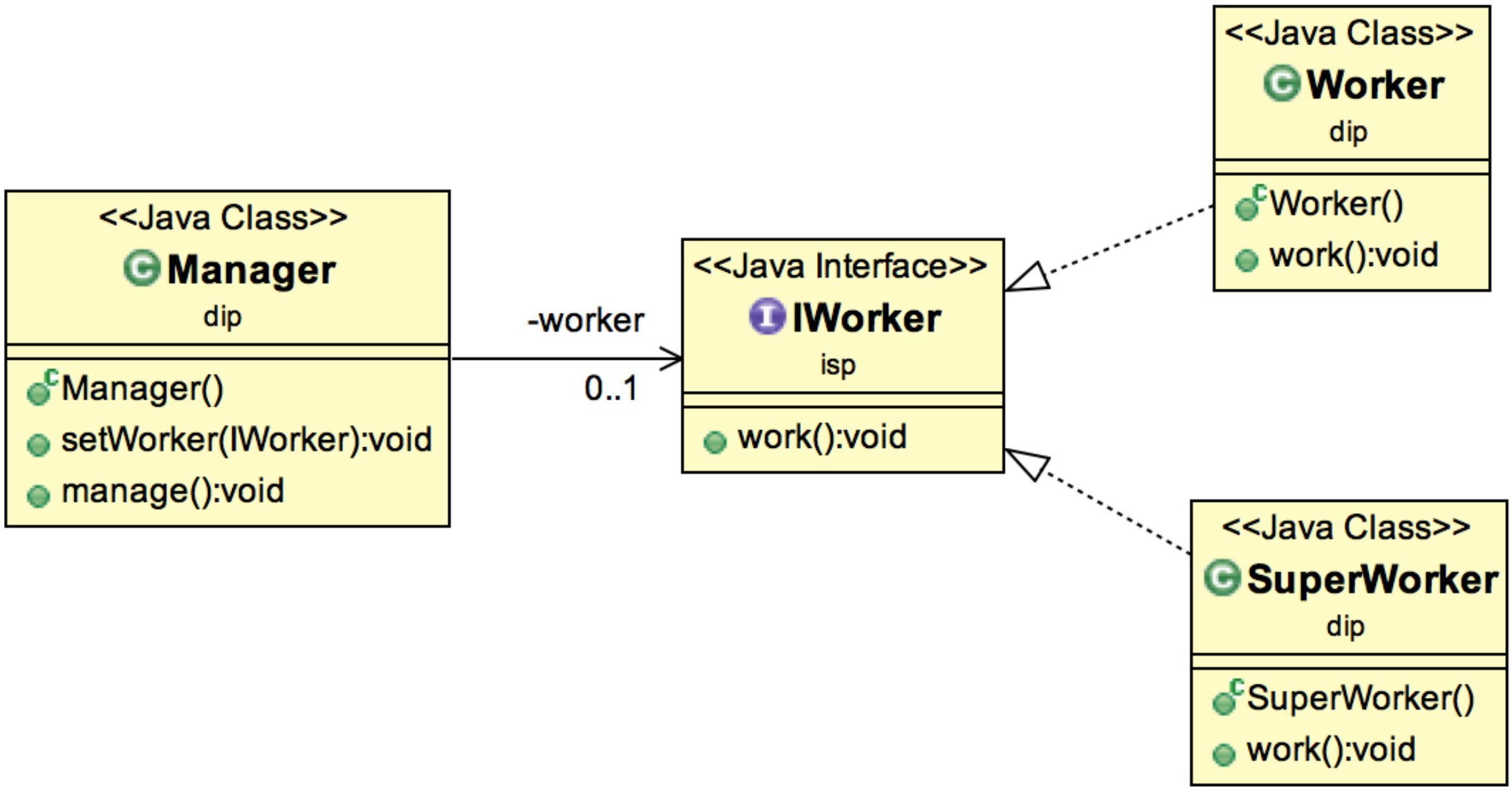

INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?

Dependency Inversion Principle *(DIP)*

Toutes entité doit dépendre d'abstraction, et non d'implémentation


<<Java Class>>
Manager
dip
Manager()
setWorker(Worker):void
manage():void
-worker
0..1
<<Java Class>>
Worker
dip
Worker()
work():void


<<Java Class>>
SuperWorker
dip
SuperWorker()
work():void
<<Java Class>>
Manager
dip
Manager()
setWorker(Worker):void
manage():void
-worker
0..1
<<Java Class>>
Worker
dip
Worker()
work():void


<<Java Class>>
Manager
dip
Manager()
setWorker(IWorker):void
manage():void
-worker
0..1
<<Java Interface>>
IWorker
isp
work():void
<<Java Class>>
Worker
dip
Worker()
work():void
<<Java Class>>
SuperWorker
dip
SuperWorker()
work():void

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware
Knowledge Software

SOLID

DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

Anti pattern

Un AntiPattern est une solution à un problème contre productive entrainant des bugs, des problèmes de performance, ou une complexité de maintenance.

- Object orgy
- God class
- Poltergeist

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

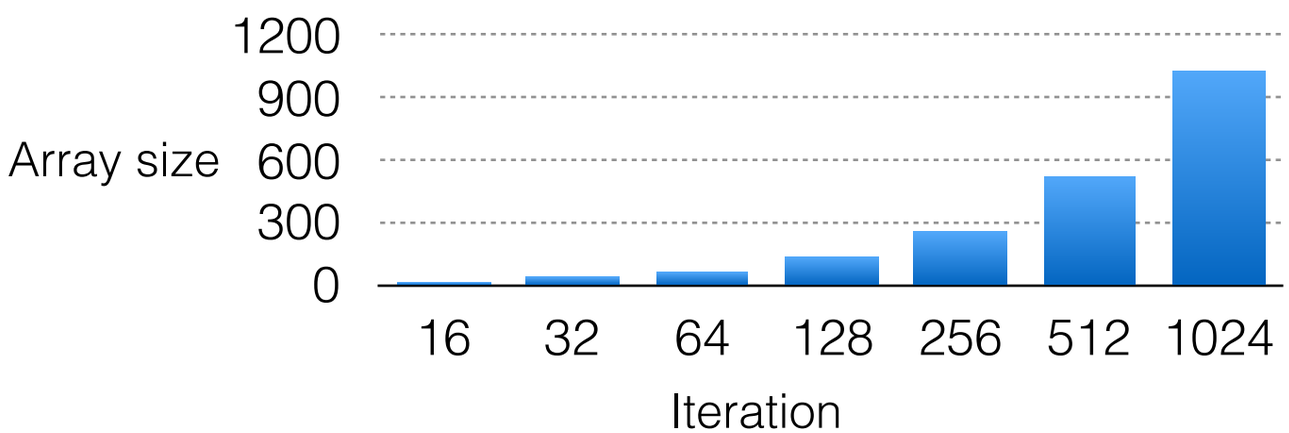
Kware
Knowledge Software

Maîtrise des outils

Java API

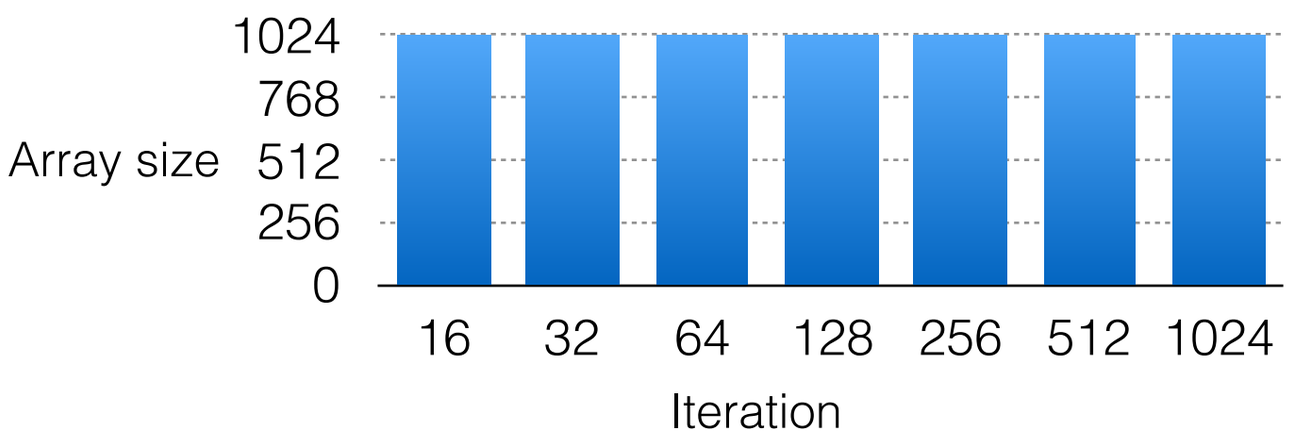
Bien connaître les API standards de **Java** permet d'optimiser les performances :

```
final List<Integer> list = new ArrayList<>();
for (int i = 0; i < 1024; i++) {
  list.add(i);
}
```



Bien connaître les API standards de **Java** permet d'optimiser les performances :

```
final List<Integer> list = new ArrayList<>(1024);
for (int i = 0; i < 1024; i++) {
  list.add(i);
}
```



StringBuilder

Bien connaître les API standards de **Java** permet d'optimiser les performances :

```
final String text = "foo" + "bar" + "coin";
```

```
final StringBuilder builder = new StringBuilder();
builder.append("foo");
builder.append("bar");
builder.append("coin");
final String text = builder.toString();
```

Static factory

Utiliser des méthodes statiques de construction d'objet plutôt que des constructeurs classiques :

```
class Complex {

    public static Complex fromCartesian(double real, double imag) {
        return new Complex(real, imag);
    }

    public static Complex fromPolar(double modulus, double angle) {
        return new Complex(modulus * cos(angle), modulus * sin(angle));
    }

    private Complex(double a, double b) {
        //...
    }
}
```

Object

La classe **Object** est conçu pour l'extension.

- equals
- hashCode
- toString

Object#equals

La méthode ***equals(Object other)*** doit respecter le contrat suivant :

- Reflexive
- Symétrique
- Transitive
- Consistant
- Non **null**

Object#equals

La méthode ***hashCode()*** doit être implémentée si **equals(Object other)** l'est.

- Consistant
- Deux objets égaux doivent avoir le même hash code
- Deux objets non égaux ne doivent pas avoir de hash code différent

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware
Knowledge Software

Couverture

LoC

Couverture

Exprimé en pourcentage, la couverture de code représente la proportion de code couvert par l'execution d'une suite de test.

100% LoC <=> L'ensemble du code est testé

Le nombre **cyclomatique** ou complexité **cyclomatique** est une métrique pour mesurer la complexité d'une classe / méthode.

$$M = E - N + 2P$$

- E = Nombre d'arêtes du graphe
- N = Nombre de noeuds du graphe
- P = Nombre de composantes connexes du graphe

Le taux de couverture à atteindre peut être lié directement au nombre **cyclomatique** :

- Plus la complexité est élevé plus la couverture doit être importante.
- Plus la complexité est faible moins la couverture est pertinente.

D’autres métriques permettent d’évaluer la qualité du code :

- Nombre de ligne de code
- Nombre de ligne de code par entité (classe, méthode, etc …)
- Nombre d’entité
- Complexité des dépendances
- Nombre de paramètre par méthode

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware
Knowledge Software

Evaluation

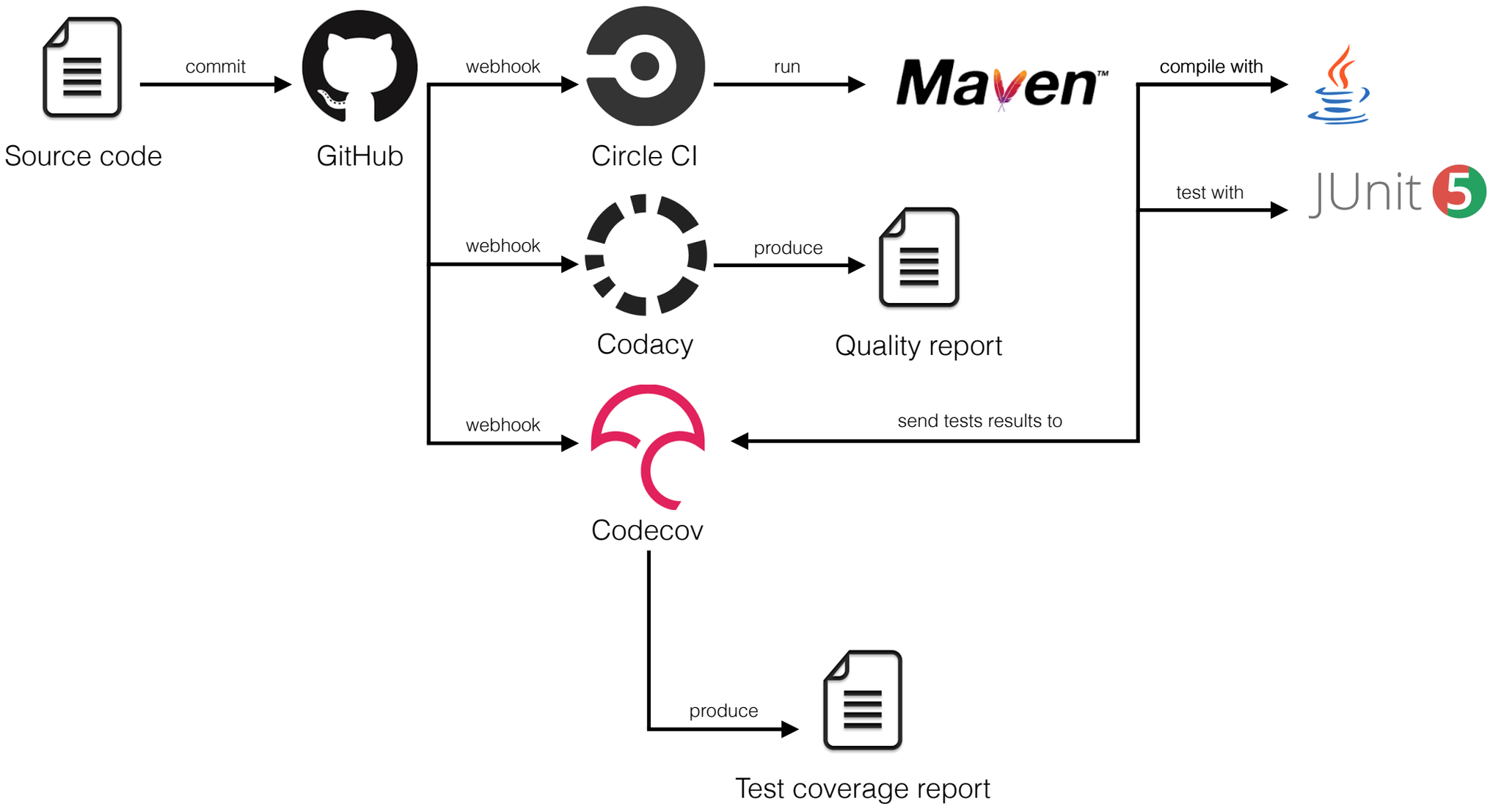
Codacy

Outils d’évaluation de la qualité de code :

- FindBugs
- PMD
- CheckStyle

Le meilleur outil reste la revue de code !

Introduction



Codacy

Codacy est une plateforme cloud offrant les services suivants :

- Analyse de code statique
- Couverture de code
- Duplication
- Complexité

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware Knowledge Software

Codacy

B

Project Certification

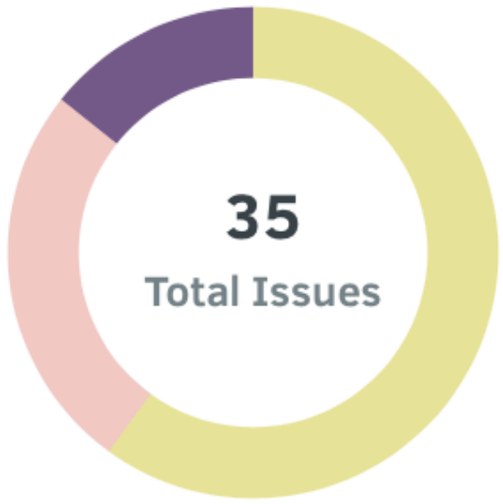
Code Complexity No Patterns	**Code Style** 83%
Compatibility 100%	**Documentation** No Patterns
Error Prone 55%	**Performance** 100%
Security 100%	**Unused Code** 43%

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware Knowledge Software

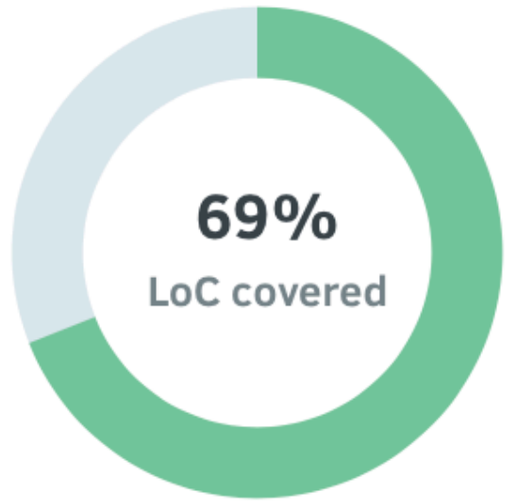
Codacy

Issues Breakdown



Unused Code	21
Code Style	9
Error Prone	5

Coverage



FILES WITH LEAST COVERAGE

list_screen.py	0%
select_screen.py	35%
input_screen.py	57%

View All

Codacy

```
145         :param surface: Optional surface this flow will be rendered into.
146         """
147         self._screens = {}
148         self._factories = {}
149         self._style_factory = StyleFactory()
150         self._font_manager = FontManager()
151         configure_screenflow(self)
152         self._running = False
153         self._stack = []
154         self._state = ScreenFlow.CREATING
155         self._surface = surface
156         self._transition = None
157 
158     @property
159     def surface(self):
160         """Surface property getter. If current surface is None,
161         it creates a fullscreen surface.
162 
163         :returns: Target surface.
164         """
165         if self._surface is None:
166             logger.info('Target surface not specified')
167             logger.info('Creating surface')
168             info = Info()
169             resolution = (info.current_w, info.current_h)
170             flags = FULLSCREEN | HWSURFACE | DOUBLEBUF
171             logger.info('Creating surface (%s, %s)' % resolution)
172             self._surface = set_mode(resolution, flags)
173         return self._surface
```

```
69
70  import logging
71  import xmltodict
```

Unused time imported from pygame

```
72  from pygame import time, FULLSCREEN, HWSURFACE, DOUBLEBUF
73  from pygame.display import set_mode, flip, Info
74  from screens import configure_screenflow
```

UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

Kware Knowledge Software

Introduction