

# AGILE GALAXY

The SCRUM MASTER

# Une Terre d'explorateur



D'un côté un univers avec des astronautes et des fusées..

Un univers d'espoir



... De l'autre des  
héros de la Force

# Pour la liberté d'expression !

```
public abstract class AbstractIndividuAlignement {  
    public abstract String sePreparer();  
  
    public abstract String agir();  
  
    public String faireDuSport() {  
        System.out.println("Fais un footing pour s'exercer");  
        return "Fais un footing pour s'exercer";  
    }  
}
```

On créer un état pour l'alignement Obscur / Lumineux.

On décide que la manière de faire du sport ne devrait pas forcément changer en fonction de l'alignement.

# Pour la lumière !

```
public class IndividuAlignementLumineux extends AbstractIndividuAlignement

    @Override
    public String sePreparer() {
        System.out.println("fais le vide dans son esprit");
        return "fais le vide dans son esprit";
    }

    @Override
    public String agir() {
        System.out.println("utilise la meditation");
        return "utilise la meditation";
    }
}
```

# Pour l'Obscurité !

```
public class IndividuAlignementObscur extends AbstractIndividuAlignement {  
  
    @Override  
    public String sePreparer() {  
        System.out.println("fais les 100 pas comme Dark Maul");  
        return "fais les 100 pas comme Dark Maul";  
    }  
  
    @Override  
    public String agir()  
    {  
        System.out.println("lance d'eclairs tout autour de lui");  
        return "lance d'eclairs tout autour de lui";  
    }  
}
```

## On intègre l'alignement à la classe existante

```
public class Maitre extends Individu {  
  
    private AbstractIndividuAlignement alignement;  
    private int karma = 0;  
  
    public Maitre(String nom, String prenom, String faction) {  
        super(nom, prenom, faction);  
        alignement = new IndividuAlignementLumineux();  
    }  
}
```

# On sécurise le changement d'alignement

```
public synchronized void setKarma(int karma) {  
    this.karma = karma;  
    if (karma < 0) {  
        alignement = new IndividuAlignementObscur();  
        this.setFaction("Obscur");  
    } else {  
        alignement = new IndividuAlignementLumineux();  
        this.setFaction("Jedi");  
    }  
}
```



On a plus qu'à déléguer les appels à l'état courant

```
@Override
public String agir() {
    return alignement.agir();
}

@Override
public String faireDuSport() {
    return alignement.faireDuSport();
}

@Override
public String sePreparer() {
    return alignement.sePreparer();
}
```

# On écrit ensuite nos exigences ...

```
» Feature: Le comportement du maitre varie en fonction de son alignement

» Scenario: Quand un maitre choisi le cote de la force il doit pouvoir mediter calmement
  Given Un maitre bon
  When Quand le maitre bon attaque
  Then Elle/il medite

» Scenario: Quand un maitre rejoint le cote obscur il doit pouvoir lancer des eclairs
  Given Un maitre mauvais
  When : Quand le maitre mauvais attaque
  Then Elle/il lance des eclairs
```

.. Qu'on implémente tout de suite après

```
private Maitre maitreGentil;  
private Maitre maitreMechant;  
private String msgMaitreGentil;  
private String msgMaitreMechant;  
  
@Given("^Un maitre bon$")  
public void un_maitre_bon() throws Exception {  
    this.maitreGentil = new Maitre("Kenobi", "Obiwan", "Jedi");  
}  
  
@When("^Quand le maitre bon attaque$")  
public void quand_le_maitre_bon_attaque() throws Exception {  
    this.msgMaitreGentil = this.maitreGentil.agir();  
}  
  
@Then("^Elle/il medite$")  
public void elle_il_medite() throws Exception {  
    assertEquals(msgMaitreGentil, "utilise la meditation");  
}
```

.. Qu'on implémente tout de suite après

```
@Given("^Un maitre mauvais$")
public void un_maitre_mauvais() throws Exception {

    this.maitreMechant = new Maitre("Skywalker", "Anakin", "Jedi");
    this.maitreMechant.setKarma(-5);
}

@When("^: Quand le maitre mauvais attaque$")
public void quand_le_maitre_mauvais_attaque() throws Exception {
    this.msgMaitreMechant = this.maitreMechant.agir();
}






@Then("^Elle/il lance des eclairs$")
public void elle_il_lance_des_eclairs() throws Exception {
    assertEquals(this.msgMaitreMechant, "lance d'eclairs tout autour de lui");
}
```

# Tout fonctionne !

Runs: 4/4

Errors: 0

Failures: 0

- ▼  dauphine.agile.starwars.StateTest [Runner: JUnit 4] (0,004 s)
  -  verificationComportementAgirMaitre (0,004 s)
  -  verificationComportementSportMaitre (0,000 s)
  -  verificationComportementSepreparerMaitre (0,000 s)
  -  verificationComportementIndividu (0,000 s)

# Jedi mais pas pilote



Une montée en  
compétence  
s'impose !

La NASA à la  
rescousse !

# Vite il nous faut plus de concombres !

---

**Feature:** On veut pouvoir qu'un utilisateur de la force puisse piloter un vaisseau spatial

**Scenario:** A la suite de cette formation, le maitre Jedi obtient un adaptateur qui va lui permettre de piloter

**Given** Un maitre

**When** Après sa formation au sein de la NASA

**Then** Le maitre peut conduire le vaisseau spatial

**Scenario:** A la suite de cette formation, le padawan obtient un adaptateur qui va lui permettre de piloter la

**Given** Un padawan

**When** Après sa formation padawan au sein de la NASA

**Then** Le padawan peut conduire le vaisseau spatial

# La NASA à la rescousse !

```
public class IndividuAstronautAdapter implements IAstronaut
{
    private final IIndividu individu;
    private String state = "calm";

    public IndividuAstronautAdapter(IIndividu individu) {
        super();
        this.individu = individu;
    }

    @Override
    public void piloter(Rocket rocket)
    {
        individu.sePreparer();
        individu.agir();
        rocket.takeOff();
    }
}
```

Un bel adaptateur  
pour former les  
individus utilisateurs  
de la Force au  
pilotage de fusée.



# Des astronautes ++

Ces utilisateurs de la Force sont même plus résistants que les autres !

```
@Override
public boolean changeState(String newState) {
    if("sick".equals(newState)) {
        System.out.println("un individu utilisateur de la force n'est jamais malade !");
        return false;
    } else {
        this.state = newState;
        return true;
    }
}
```

Update

# Un centre de certifications officiel

La NASA fait une static method factory pour délivrer ses adaptateurs !




```
public class CentreDeCertification
{
    public static IAstronaut delivrer(IIndividu individu)
    {
        return new IndividuAstronautAdapter(individu);
    }
}
```

# Les certifications sont prêtes !

Runs: 2/2

✖ Errors: 0

✖ Failures: 0

- ▼  dauphine.agile.fusion.AdaptateurTest [Runner: JUnit 4] (0,073 s)
  -  insertionMaitreAvecCertificat (0,070 s)
  -  insertionPadawanAvecCertificat (0,003 s)

C'est la fin de la formation et le début des aventures !

