Classe Card

On modélise ici une carte à jouer. On ira par touches successives en introduisant les concepts Java et Objet. Vous devriez pouvoir suivre le cours sans connaissances particulières en Java. Pour chaque notion, on renverra aux parties du tutoriel Oracle concerné.

Configuration du projet

On suppose que vous avez crée un nouveau projet en suivant le tutorial, donc que les paramètres de votre projet sont :

- nouveau projet avec *Java 1.8*
- on choisit comme template : *Command Line Application*
- Package de base `com.mco`

Créez des classes avec Intellij et la fenêtre projet

Pour voir la fenêtre projet cliquez sur : **"view » Tool Windows » project"** ceci cachera le projet s'il était visible et le montrera s'il était caché. Puis pourvoir vos fichiers :

- cliquez sur le petit triangle à côté nom du projet,
- cliquez sur "src" (acronyme pour sources) qui contient vos fichiers.
- vos fichiers sont rangés par package. Vous devriez donc voir une classe Main dans le package com.mco

Lorsque l'on crée une nouvelle classe, il faut la placer dans le bon package. Cliquez donc **toujours** sur le bon package avant de créer une nouvelle classe avec le clique droit ou le menu file.

Plan

On procèdera par touches successives :

- v1 : Propriétés minimales d'une carte
  - v1.0 : uniquement les attributs et sans se soucier de packages et de visibilité.
  - v1.1 : on règle les problèmes de visibilité (changement de package et accesseurs publics)
- v2 : on peut afficher à l'écran une carte !
- v3 : on remplace l'attribut couleur par un enum
- v4 : la difficile question de l'égalité.
- v5 : un constructeur par défaut.

Card v1

On commence par créer une classe avec uniquement des attributs et un constructeur, la v1.0 puis on

ajoute les contrainte d'encapsulation des attributs.

Card v1.0

Notions vues :

- **attributs** leurs valeurs va différentier un objet de cette classe d'un autre.
- **constructeur** ce qui va créer un objet de la classe.
- **visibilité** par défaut le package. Si tout se passe comme prévu on changera

Pour créer une classe (et le fichier qui va avec) :

1. placez vous dans le bon package en cliquant dessus dans la fenêtre projet,
2. **"File » New » Java class"**

Une fois dans le bon package on peut aussi :

- **"clic droit » New » Java class"**
- **"CTRL + N"**

Card.java

```
package com.mco;

public class Card {
    int value;
    String color;

    Card(int value, String color) {
        this.value = value;
        this.color = color;
    }
}
```

Faites du code **LISIBLE** :

- noms de variable explicatifs,
- code bien indenté (**"Code » Reformat code"** ou Ctrl+Alt+L par défaut peut vous aider)

Main.java

On teste notre code :

- un code non testé est un code cassé,
- on va garder nos tests pour pouvoir les exécuter à chaque modification de code.

Pour la V1, les attributs de visibilité nous permettent d'utiliser les champs directement :

Notions vues :

- **mot clé static**
- **Exécution de programme**

```
package com.mco;

public class Main {

    private static void testCreationCard() {
        Card septDeCoeur = new Card(7, "coeur");

        System.out.println(septDeCoeur.value);
        System.out.println(septDeCoeur.color);
    }

    public static void main(String[] args) {
        testCreationCard();
    }
}
```

Le résultat devrait être :

```
7
coeur
```

Avec Intellij, ce que l'on exécute avec le menu **"run » run"** ou le triangle vert en haut à droite de la fenêtre est déterminé par le nom à coté du triangle vert. Pour nous il devrait y avoir marqué `Main`. Pour changer ce qui est exécuté (par exemple si l'on a changé la classe Main de package) :

- cliquez sur le triangle pointant vers le bas à côté du texte Main
- cliquez sur **"Edit Configurations..."**
- vous devriez ouvrir une nouvelle fenêtre avec
  - à gauche une liste de noms et Main en surbrillance
  - à droite une liste d'options. Celle qui nous interesse est "Main class:" dans l'onglet configuration (qui est sélectionné par défaut).

La ligne **"Main class:"** de l'onglet configuration doit contenir `com.mco.Main`, c'est la

classe qui est exécutée. Si vous avez changé la classe Main de package ou que ce n'est plus ça que vous voulez exécuter, placez le bon nom de la classe ici.

v1.1 : visibilité

On utilise quelques règles :

- le code est placé dans des packages particuliers,
- les attributs sont sans modificateur (setter) mais accessibles via des accesseurs (getters),
- les méthodes constructeurs sont en `public`

Notions vues :

- **visibilité**
- **base des packages**
- accéder aux attributs de l'extérieur via des getters/setters
- design pattern : value object

Changement de package

Changer le package de `Card.java` doit se faire en changeant de dossier (packages et dossiers sont identiques). Pour que cela se fasse sans douleur, on utilise les possibilités de l'IDE :

1. on change le nom du package de `Card.java` : le code devient rouge, package et répertoire ne coïncident plus
2. on se place sur le rouge et on clique sur l'ampoule qui propose des moyens de corriger de problème
3. on choisit `move to package com.mco.battle`

Le code ne marche plus. On le corrige avec des `getters` (**"Code » Generate… » getter/setter"**).

On ne veut pas pouvoir changer la valeur de la carte une fois créée. C'est illogique en vrai et dans le code on aurait des variables `septDeCoeur` qui seraient en fait des as de pique…

Lorsque l'on aura le choix on utilisera toujours des objets **non modifiables** (on dit aussi non mutables). C'est un *design pattern* (façon de faire) commun. Il s'appelle : value object

Card.java

```
package com.mco.battle;

public class Card {
    int value;
    String color;

    public Card(int value, String color) {
        this.value = value;
        this.color = color;
    }

    public int getValue() {
        return value;
    }

    public String getColor() {
        return color;
    }
}
```

Main.java

```
package com.mco;

import com.mco.battle.Card;

public class Main {

    private static void testCreationCard() {
        Card setDeCoeur = new Card(7, "coeur");

        System.out.println(setDeCoeur.getValue());
        System.out.println(setDeCoeur.getColor());
    }

    public static void main(String[] args) {
        testCreationCard();
    }
}
```

v2 Affichage d'une carte à l'écran

Si on essaie d'afficher une carte :

```
System.out.println(new Card(7, "coeur"));
```

On obtient :

```
com.mco.battle.Card@511d50c0
```

Dès que l'on a besoin de transformer un objet en chaîne de caractères (ici la fonction `System.out.println` affiche une chaîne de caractères à l'écran), Java utilise la méthode `String toString()`.

Comme nous n'avons pas défini de méthode `toString`, c'est celle de la classe Object qui est utilisée.

Pour en faire une nous-mêmes : **"code » generate » toString"**

L'annotation `@Override` signifie que nous avons récrit une méthode d'une classe ancêtre. Par défaut toute nouvelle classe hérite d'Object qui définit les méthodes que toutes les classes doivent avoir.

Notions vues :

- transformer un objet en chaîne de caractères avec `String toString`
- première forme d'héritage : toute classe hérite de la classe Object.

Card.java

```java
package com.mco.battle;

public class Card {
    int value;
    String color;

    public Card(int value, String color) {
        this.value = value;
        this.color = color;
    }

    public int getValue() {
        return value;
    }

    public String getColor() {
        return color;
    }

    @Override
    public String toString() {
        return "Card{" +
                "value=" + value +
                ", color='" + color + '\'' +
                '}';
```

```
    }
}
</code java>

=== Main.java ===

<code java>
// snip
    private static void testCreationCard() {
        Card setDeCoeur = new Card(7, "coeur");

        System.out.println(setDeCoeur);
        System.out.println(setDeCoeur.getValue());
        System.out.println(setDeCoeur.getColor());
    }

/snip
```

v3 : les couleurs comme une énumération

Passer les valeurs et les couleurs de la carte en paramètres ne semble pas être une bonne idée (code smell). De façon générale, pour gérer les constantes d'un type particulier et éviter de se tromper, on utilise en Java des Enum. En tous les cas, on n'utilise pas des chaînes de caractères comme on l'a fait jusqu'à présent. C'est un magic number et en code, c'est **MAL**.

On va le faire ici pour les couleurs, peut-être que ce sera également utile pour les valeurs, mais nous ne le ferons pas.

Notions vues :

- les Enum
- le code smell, 6ème sens du codeur
- design pattern : pas de magic number
- composition des toString
- le switch comme structure de contrôle

Colors.java

On se place dans le bon package puis **"Clic droit » New » Java Class"** Choisissez ensuite **"Enum"** comme type.

```
package com.mco.battle;

public enum Colors {
    SPADE,
    HEART,
```

```
    DIAMOND,
    CLUB;

    @Override
    public String toString() {
        switch (this) {
            case SPADE:
                return "spade";
            case HEART:
                return "heart";
            case DIAMOND:
                return "diamond";
            case CLUB:
                return "club";
            default:
                return "other";
        }
    }
}
```

Card.java

Après avoir changé le type de `color`, on peut supprimer les *getters* et le `toString` pour les regénérer.

```
package com.mco.battle;

public class Card {
    int value;
    Colors color;

    public Card(int value, Colors color) {
        this.value = value;
        this.color = color;
    }

    public int getValue() {
        return value;
    }

    public Colors getColor() {
        return color;
    }

    @Override
    public String toString() {
        return "Card{" +
                "value=" + value +
                ", color=" + color +
```

```
                '}';
    }
}
```

Main.java

Il n'y a qu'à la création de la carte qu'il faut toucher. Le reste fonctionne toujours.

```
// snip
Card setDeCoeur = new Card(7, Colors.HEART);
//snip
```

v4 : Egalité entre cartes

En java l'opérateur == sert à deux choses :

- tester l'égalité pour les types primitifs (types sans majuscule : nombres et booléens en gros)
- tester l'égalité des objets. Si ce sont les **mêmes** ou pas.

Exemple du code ci-dessous exécuté dans le main :

```
    private static void testEqualityCard() {
        Card asDePique = new Card(1, Colors.SPADE);
        System.out.println(asDePique == asDePique);
        System.out.println(asDePique == new Card(1, Colors.SPADE));
    }
```

Pour l'égalité de contenu, on utilise la méthode `equals` définie dans la classe Object qu'il faut (comme `toString`) regénérer ("**Code » Generate » equals() and hashcode()**").

On peut voir ces méthodes comme des méthodes de conversion d'objets :

- `String toString()` : convertit un objet en chaîne de caractères,
- `boolean equals(Object o)` : convertit en booléen si le **contenu** est le même qu'un autre
- `int hashcode()` : convertit un objet en nombre.

On ne génère **jamais** `equals` tout seul. On lui associe toujours `hashcode` car deux objets égaux avec equals doivent avoir le même hashcode.

C'est à savoir si vous faites vous-même ces méthodes pour vos objets.

Notions vues :

- les types primitifs
- égalité entre objets (la note)
- les méthodes equals et hashcode.

Card.java

On laisse Intellij générer les méthodes. C'est un peu sale mais ça fait le job. Regardez http://www.ideyatech.com/effective-java-equals-and-hashcode/ qui explicite les façons de faire décrites dans ce magnifique livre qu'est effective Java.

Main.java

```
//snip
    private static void testEqualityCard() {
        Card asDePique = new Card(1, Colors.SPADE);
        System.out.println(asDePique == asDePique);
        System.out.println(asDePique == new Card(1, Colors.SPADE));
        System.out.println(asDePique.equals(new Card(1, Colors.SPADE)));
        System.out.println(new Card(1, Colors.SPADE).equals(asDePique));
    }
//snip
```

v5 : Constructeur par défaut

Moins il y a de paramètres à une méthode, mieux c'est. Uncle Bob dans son livre clean code dit que le meilleur nombre de paramètres pour une méthode est 0.

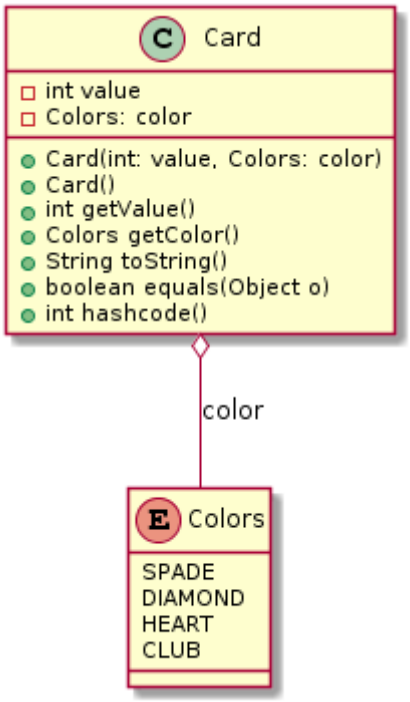
Notions vues :

- minimiser le nombre de paramètres d'une méthode
- mot clé this

Card.Java

```
//snip
    public Card() {
        this(1, Colors.SPADE);
    }
//snip
```

Diagramme UML final




```
enum Colors {
  SPADE
  DIAMOND
  HEART
  CLUB
}

class Card {
-int value
-Colors: color
+Card(int: value, Colors: color)
+Card()

+int getValue()
+Colors getColor()
+String toString()
+boolean equals(Object o)
+int hashcode()
}

Card o-- Colors : color
```