Classes et records

Tristan Crolard

Laboratoire CEDRIC
Equipe « Systèmes Sûrs »

tristan.crolard@cnam.fr

cedric.cnam.fr/sys/crolard

Valeurs et Objets

- Tout est **objet** et un objet possède une **identité propre**
  (penser aux objets du monde réel : chaise, table ...)
- Pas de réelle notion de **valeur** en Java sauf pour les types atomiques.
- Par exemple, la valeur 42, de type `int` n'a pas de d'identité propre.
- Plus généralement, dans le monde mathématique, tout est **valeur**.

- En java, pour chaque classe qui définit des valeurs, il faut donc :
  - s'interdire d'utiliser l'égalité primitive "=="
  - redéfinir l'égalité `equals` (et n'utiliser que celle-là)
  - redéfinir la fonction `hashCode` (qui doit être toujours compatible avec `equals`)

Remarque : les IDE peuvent générer ces fonctions automatiquement.

Exemple : la classe `Pair`

```
import java.util.Objects;


public class Pair<A, B> {
    final A fst;
    final B snd;

    public Pair(A fst, B snd) {
        this.fst = fst;
        this.snd = snd;
    }

...
```

Exemple : hashCode et equals

...

```
    @Override
    public int hashCode() {
        return Objects.hash(fst, snd);
    }

    @Override
    public boolean equals(Object obj) {
        if (obj instanceof Pair<?, ?>) {
            Pair<?, ?> other = (Pair<?, ?>)obj;
            return (this.fst.equals(other.fst) &&
                    this.snd.equals(other.snd));
        }
        return false;
    }
}
```

Les records

Introduits en Java 14, ils permettent de définir facilement des types de valeurs.

Exemple

```
record Pair<A,B>(A fst, B snd)
```

Sont définis automatiquement :

- `toString()`
- `hashCode()` et `equals()`
- les accesseurs
- un constructeur publique