M2103 (POO)
Bases de la programmation orientée objet

Responsable : Cristina Onete
cristina.onete@gmail.com
http://onete.net/#teaching

Avec les contributions d'Isabelle Blasquez et Marin Bertier

Récapitulation

- La notion de **this** :

```
public class Nombre {
        private int valeur;

        public Nombre(int valeur){
                this.valeur = valeur;
        }
}
```

```
public class Etudiant {
        private String nom;
        private String prenom;

        public Etudiant(String nom, String prenom){
                this.nom = nom;
                this.prenom = prenom;
        }
}
```

Récapitulation -- 2

- Les constructeurs

```
public class Nombre {
        private int valeur;

        public Nombre(int valeur){
                this.valeur = valeur;
        }
}
```

```
public class MonCode {
        public static void main(String[] args){
                final Nombre n1 = new Nombre(12);
                final Nombre n2 = new Nombre();
        }
}
```

appelle le constructeur

essaie d'appeller le constructeur mais pas de constructeur avec la signature Nombre()

ERREUR DE COMPILATION

Récapitulation -- 3

- Le constructeur par défaut

```
public class Nombre {
        private int valeur;
}
```

```
public class MonCode {
        public static void main(String[] args){
                final Nombre n2 = new Nombre();
        }
}
```

Cherche un constructeur avec signature Nombre()

Ne le trouve pas...

Mais si on n'a pas de constructeur (du tout) dans la classe
ALORS un constructeur par defaut avec la signature Nombre() lui est assigné

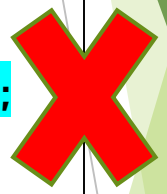
Attention : une fois qu'on rajoute un constructeur à la classe Nombre, on n'aura plus le droit à un constructeur par défaut !

Récapitulation -- 3

- Le constructeur par défaut

```
public class Nombre {
        private int valeur;

        public Nombre(int valeur){
                this.valeur = valeur;
        }
}
```

```
public class MonCode {
        public static void main(String[] args){
                final Nombre n2 = new Nombre();
        }
}
```



Récapitulation -- 4

- Plusieurs constructeurs

```
public class Etudiant {
        private String nom;
        private String prenom;
        private int annee;

        public Etudiant(String nom, String prenom, int annee){
                this.nom = nom;
                this.prenom = prenom;
                this.annee = annee;
        }

        public Etudiant(String nom, String prenom){
                this(nom, prenom, 1);
        }
}
```

Regles de fonctionnement : constructeurs

- même nom que la classe
- pas deux constructeurs avec la même signature

  Etudiant(String nom, String prenom)
  Etudiant(String prenom, String nom)

  ont la même signature

- BP 1 : le constructeur avec les plus de paramètres appellé par les autres constructeurs
- BP 2 : on met les valeurs en entrée en même ordre que les attributs et on leur donne les mêmes noms

Récapitulation -- 5

- Public vs Private, méthodes **get** (getters) :

```
public class Bloc {
        private String nom;
        private String couleur;

        public Bloc(String nom, String couleur){
                this.nom = nom;
                this.couleur = couleur;
        }
}
```

```
public class MonCode{
        public static void main (String[] args){
                final Bloc b1 = new Bloc("cube", "bleu");
                final Bloc b2 = new Bloc("etoile", "rouge");

                // verifier s'ils ont la meme couleur
                if (b1.couleur.equals(b2.couleur)){
                        System.out.println("Meme couleur !");
                }
                else {
                        System.out.println("Couleur differente !");
                }
        }

}
```

Erreur de compilation
Pourquoi ?

Récapitulation -- 5

- Public vs Private, méthodes **get** (getters) :

```
public class Bloc {
        private String nom;
        private String couleur;

        public Bloc(String nom, String couleur){
                this.nom = nom;
                this.couleur = couleur;
        }
}
```

```
public class MonCode{
        public static void main (String[] args){
                final Bloc b1 = new Bloc("cube", "bleu");
                final Bloc b2 = new Bloc("etoile", "rouge");

                // verifier s'ils ont la meme couleur
                if (b1.couleur.equals(b2.couleur)){
                        System.out.println("Meme couleur !");
                }
                else {
                        System.out.println("Couleur differente !");
                }
        }

}
```

Nous sommes dans la classe MonCode
Donc pas dans la classe Bloc

Les attributs de Bloc sont privés !

Récapitulation -- 5

- Public vs Private, méthodes **get** (getters) :

```
public class Bloc {
        private String nom;
        private String couleur;

        public Bloc(String nom, String couleur){
                this.nom = nom;
                this.couleur = couleur;
        }

        public String getCouleur(){
                return this.couleur;
        }
}
```

```
public class MonCode{
        public static void main (String[] args){
                final Bloc b1 = new Bloc("cube", "bleu");
                final Bloc b2 = new Bloc("etoile", "rouge");

                // verifier s'ils ont la meme couleur
                if (b1.getCouleur().equals(b2.getCouleur())){
                        System.out.println("Meme couleur !");
                }
                else {
                        System.out.println("Couleur differente !");
                }
        }

}
```

Solution : des méthodes get !

Récapitulation -- 6

- La notion de null
  - null veut dire : faire une référence vers une location nulle (vide)
  - String monString = null;

- Comment l'utiliser :
  - null n'est pas un objet commun
  - Si on veut chercher si qq chose est null :
    - if (monString == null) ← permis pour les String seulement pour null
    - C'est parce que on ne cherche pas savoir si la chaine de caractères stockée dedans est égale à celle stockée dans l'objet null
    - En fait, on cherche savoir si une valeur de ce String existe.
    - Attention, "" n'est pas équivalent à null !

Récapitulation -- La méthode toString()

- Son rôle : rendre une description en texte de l'objet
  - Signature String toString()
  - À la fin : return <objet de type String>

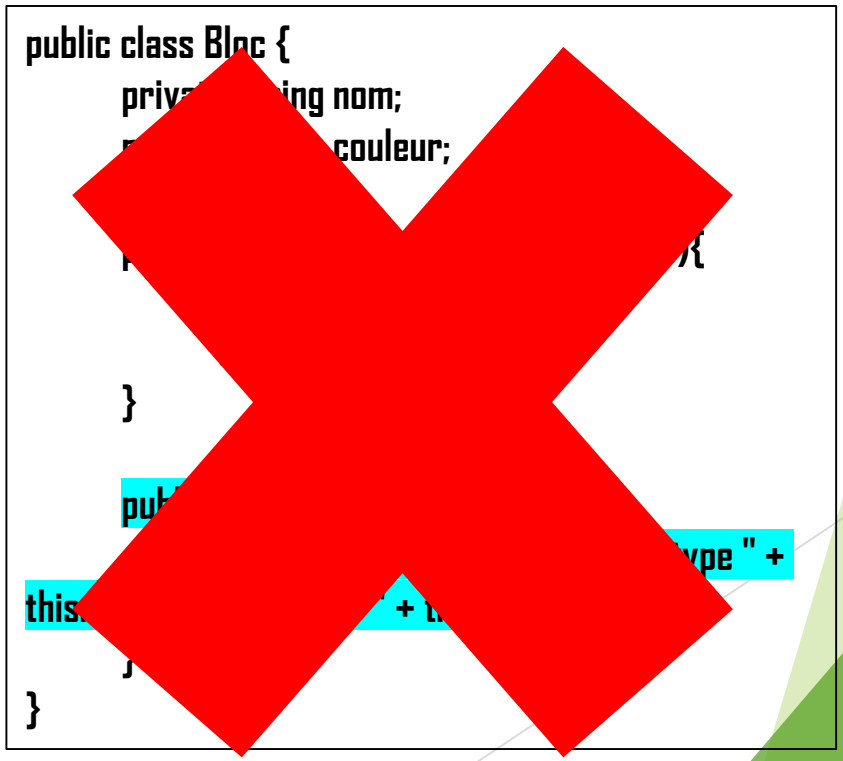
```
public class Bloc {
	private String nom;
	private String couleur;

	public Bloc(String nom, String couleur){
		this.nom = nom;
		this.couleur = couleur;
	}

	public String toString(){
		return ("Un bloc de type " + this.nom + " de couleur " + this.couleur);
	}
}
```

```
public class Bloc {
	private String nom;
	private String couleur;

	public Bloc(String nom, String couleur){
		this.nom = nom;
		this.couleur = couleur;
	}

	public String toString(){
		System.out.println("Un bloc de type " + this.nom + " de couleur " + this.couleur);
	}
}
```

Récapitulation -- La méthode toString()

- Son rôle : rendre une description en texte de l'objet
  - Signature String toString()
  - À la fin : return <objet de type String>

```
public class Bloc {
	private String nom;
	private String couleur;

	public Bloc(String nom, String couleur){
		this.nom = nom;
		this.couleur = couleur;
	}

	public String toString(){
		return ("Un bloc de type " + this.nom + "
de couleur " + this.couleur);
	}
}
```



Récapitulation -- méthode toString()

- Utiliser la méthode toString()

Chercher dans classe Bloc, voir s'il y a une méthode String toString()

```
public class Bloc {
        private String nom;
        private String couleur;

        public Bloc(String nom, String couleur){
                this.nom = nom;
                this.couleur = couleur;
        }

        public String toString(){
                return ("Un bloc de type " + this.nom + "
de couleur " + this.couleur);
        }
}
```

```
public class MonCode{
        public static void main (String[] args){
                final Bloc b1 = new Bloc("cube", "bleue");
                final Bloc b2 = new Bloc("etoile", "rouge");

                System.out.println(b1);
        }
}
```

Un bloc de type cube de couleur bleue

Récapitulation -- méthode toString()

- Utiliser la méthode toString()

Chercher dans classe Bloc, voir s'il y a une méthode String toString()

```
public class Bloc {
        private String nom;
        private String couleur;

        public Bloc(String nom, String couleur){
                this.nom = nom;
                this.couleur = couleur;
        }
}
```

```
public class MonCode{
        public static void main (String[] args){
                final Bloc b1 = new Bloc("cube", "bleue");
                final Bloc b2 = new Bloc("etoile", "rouge");

                System.out.println(b1);
        }
}
```

Location de mémoire où b1 est stocké

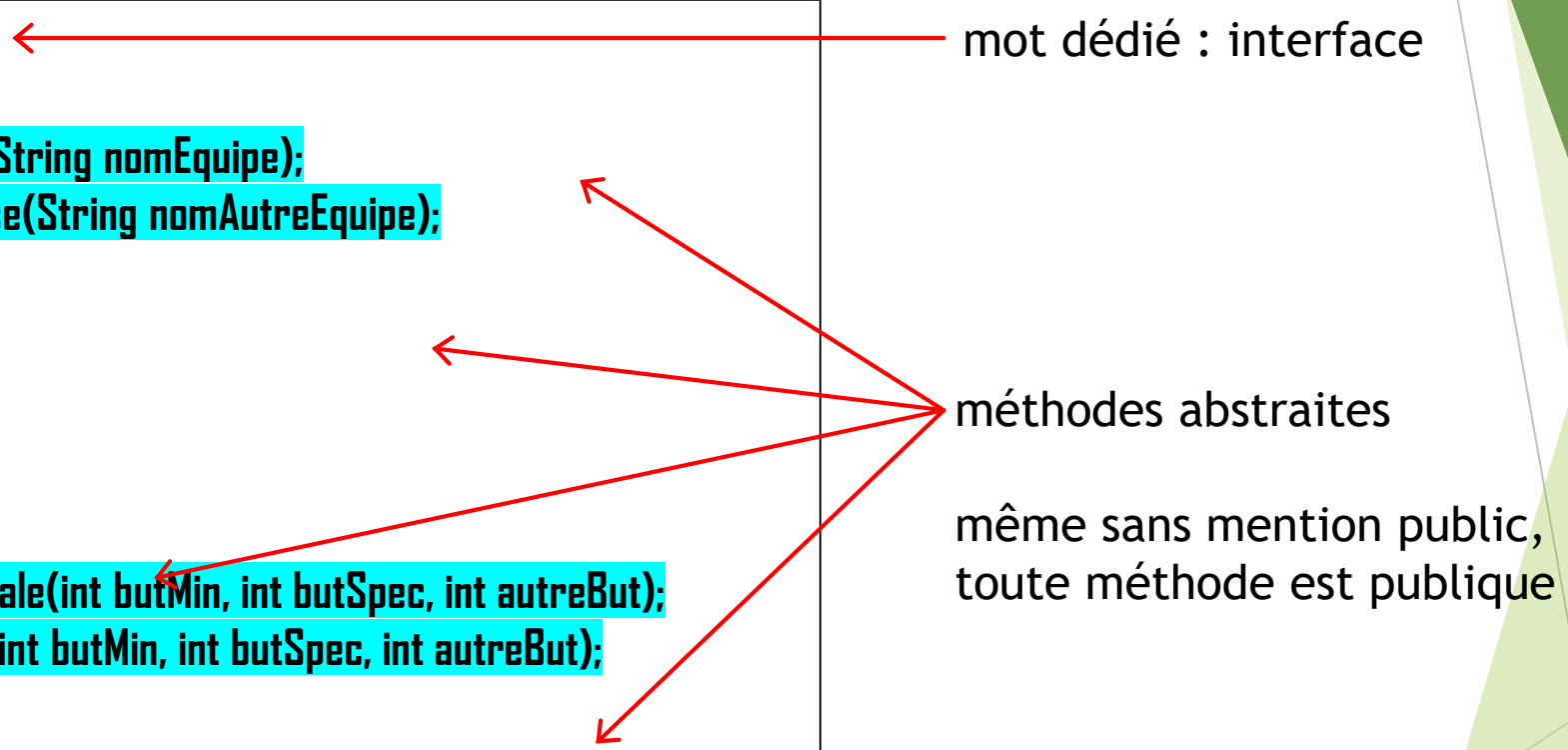
Les interfaces

Les interfaces en Java

- Une interface est un ensemble de caractéristiques liées décrites par des méthodes abstraites
  - Prenons l'exemple d'un match de sport (foot, rugby, handball, basket...)
  - Ces jeux ont des caractéristiques similaires :
    - À chaque match, il y a une équipe locale et des visiteurs
    - On a un système de points (buts, essais, penaltys, transformations)
    - On a un système de reprises, avec une durée détérminée
- Une interface Match définira ces caractéristiques en termes de méthodes
- Attention : une interface n'est pas une classe
  - Une interface est implémentée par des classes, qui concrétisent ses méthodes

L'interface Match

```
interface Match {
        // les equipes
        void equipeLocale(String nomEquipe);
        void equipeVisiteuse(String nomAutreEquipe);

        // les buts
        int butMinimal();
        int butSpeciale();
        int autreBut();

        // le score
        int scoreEquipeLocale(int butMin, int butSpec, int autreBut);
        int scoreVisiteurs(int butMin, int butSpec, int autreBut);
        String victeur();
}
```



mot dédié : interface

méthodes abstraites

même sans mention public,
toute méthode est publique

Comment utiliser cette interface

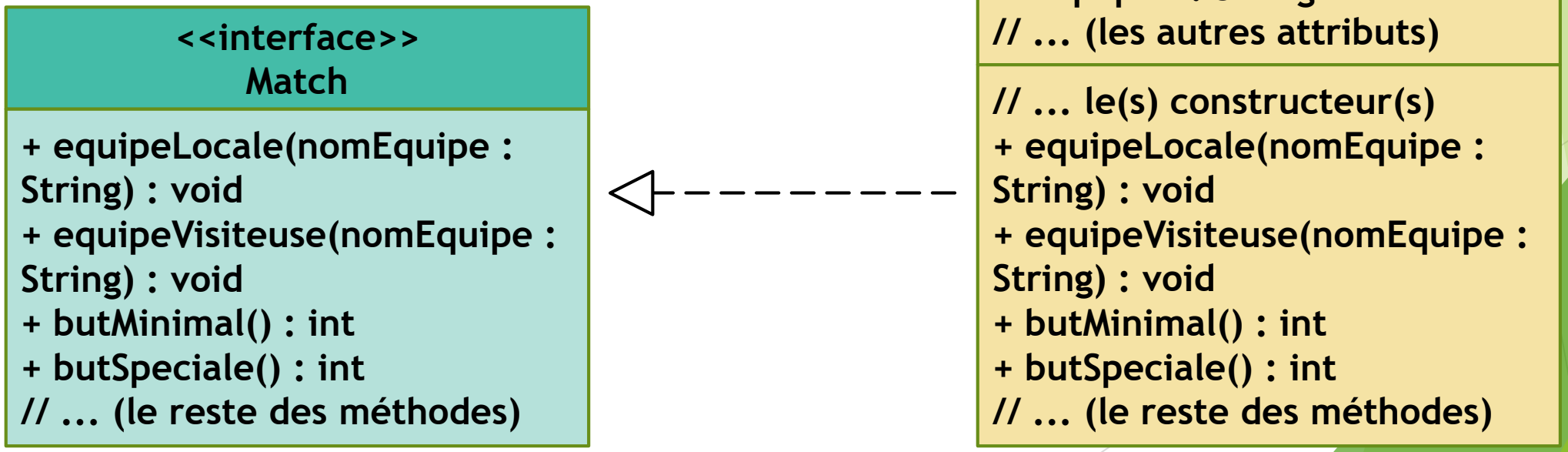
```
public class Foot implements Match {
        String equipe1, equipe2, victeur;
        int score1, score2, temps;
        // ... Un ou plusieurs constructeurs

        public void equipeLocale(String nomEquipe){
                this.equipe1 = nomEquipe;
        }
        public void equipeVisiteuse(String nomAutreEquipe){
                this.equipe 2 = nomAutreEquipe;
        }
        public void dureeDuMatch {
                this.temps = 90;
        }
        public int butMinimal(){
                return 1;}
        public int butSpeciale(){
                return 0;}
        public int autreBut(){
                return 0;}
        public int scoreEquipeLocale(int butMin, int butSpec, int
autreBut){
                this.score1 = butMin * butMinimal();
                return this.score1;}
        public int scoreVisiteurs(int butMin, int butSpec, int
autreBut){
                this.score2 = butMin * butMinimal();
                return this.score2;}

        public String victeur(){
                // ... code en détail
        }
}
```

Implémenter une interface

- Une classe qui implémente une interface détaille toutes ses méthodes
- Une interface peut être implémentée par plusieurs classes
  - Et une classe peut implementer plusieurs interfaces

```
class NomClasse implements Interface1,Interface2
```

- Diagrammes de classe

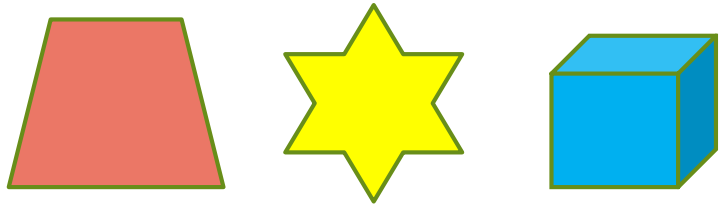


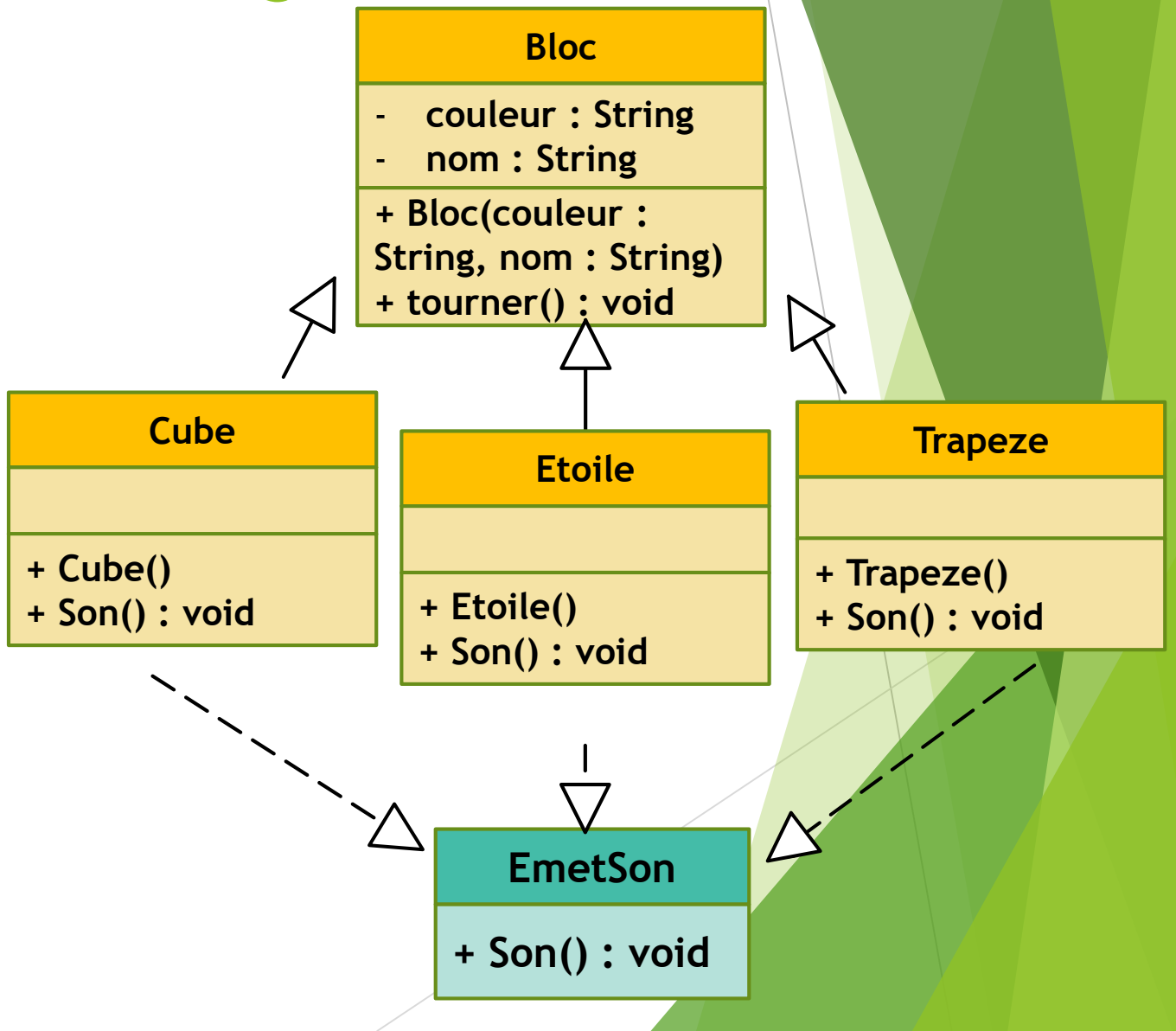
Interface vs. héritage

Héritage	Interface
• = "est un type de"	• = "fonctionne comme"
• Relation superclasse – sous-classe(s)	• Relation classe – interface
• Mot clé **extends**	• Mot clé **implements**
• Peut contenir des variables et méthodes concrètes	• Ne contient jamais de variables • N'a que des declarations de méthodes
• Une sous-classe peut redéfinir les méthodes de la superclasse • Mot clé **@Override**	• Une classe concrète doit détailler toutes les méthodes de l'interface

Exemple : interface et héritage

- On se rappelle de nos blocs :
  - Chaque bloc émet un son différent
  - Ils peuvent également être tournés



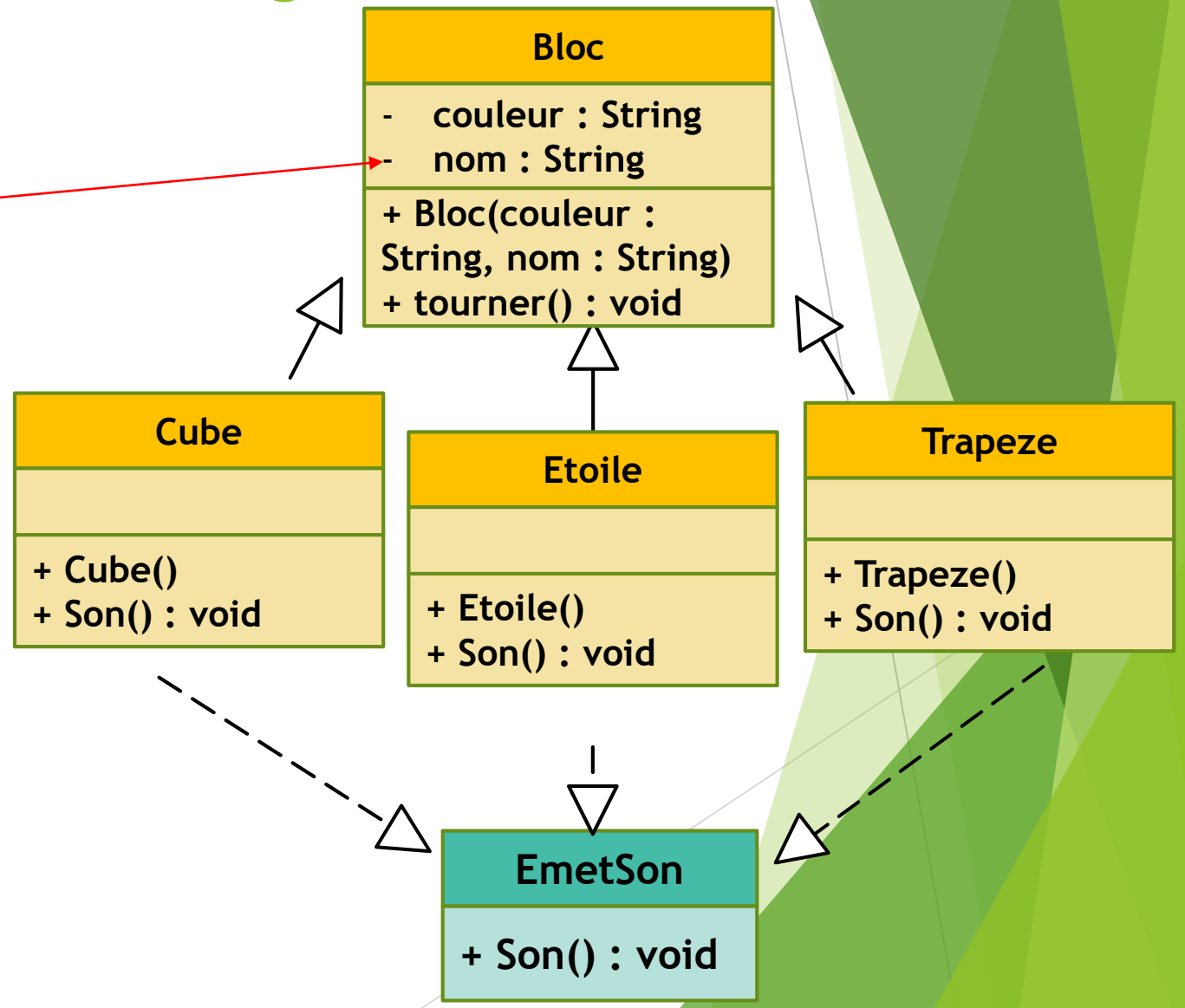


Exemple : interface et héritage

Les attributs sont privés

Alors on ne peut pas y accéder des classes Cube, Etoile, Trapeze

Il nous faut des méthodes get



Les exceptions

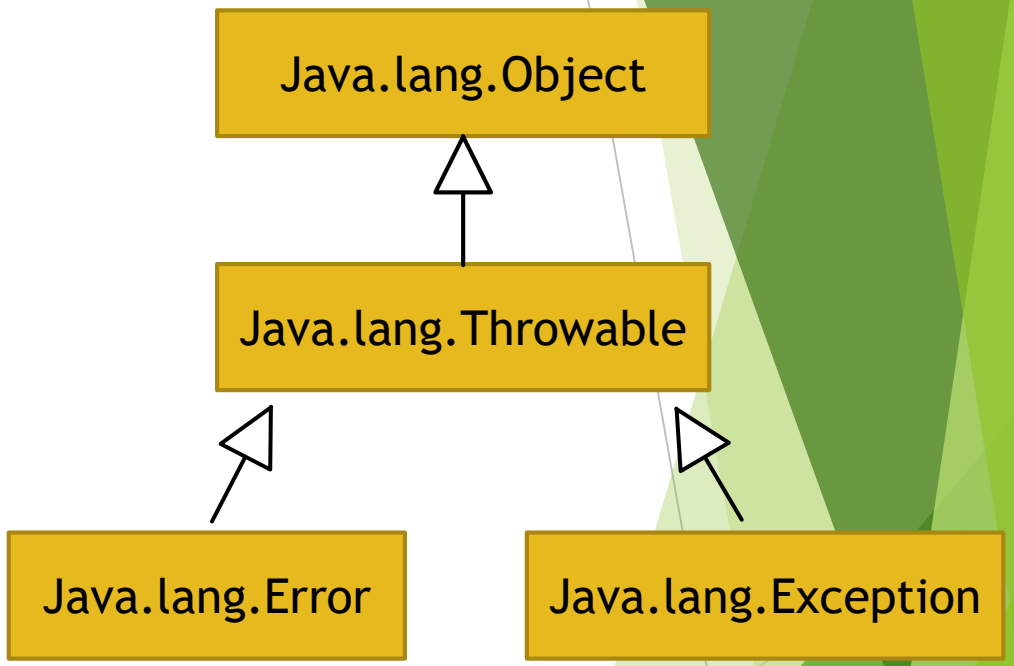
C'est quoi, une exception ?

- Une exception est un objet
- Elle est instanciée lors d'un événement qui interrompt le flot normal d'un programme
- Une exception peut apparaître lors de l'arrêt d'un programme, ou dans un comportement aberrant

Anticiper et traiter ce comportement peut éviter que des erreurs importantes se produisent

Des divers types d'exceptions

- **Error** :
  - un type d'exception terminale : il arrête l'exécution d'un programme
  - Les erreurs signalent un défaut grave et ne doivent pas être traitées (nous devons savoir si elles se produisent)
  - VirtualMachineError, OutOfMemory, ...
- **Exception :**
  - Moins grave qu'une erreur, peuvent toutefois créer des problèmes
  - Deux types : exceptions à compilation, exceptions à l'execution (runtime)
  - IOException, SQLException, NullPointerException



Les exceptions se lèvent ...

- Si une faille technique interrompt le fonctionnement du programme
  - Par exemple si un index déborde (par rapport à la taille d'un tableau)

    IndexOutOfBoundsException
  - Ou si la Java Virtual Machine rencontre un problème :

    OutOfMemoryError
- Si une faille applicative interrompt le fonctionnement du programme
  - Par exemple si un fichier donné en entrée n'est pas trouvé :

    FileNotFoundException

Comment lever des exceptions

- Mod dédié : throw
- **Lever** l'exception dans une clause if :

```
if (monFichier == null ){
        throw new FileNotFoundException("Ce fichier n'existe pas.");
}
```

- **Capturer** l'exception dans un bloc try-catch (mots dédiés)

```
try{
        // code a executer
}
catch (FileNotFoundException e) {
        // code a executer si l'exception est levee
}
```

Contrôler les exceptions

- Une exception peut être contrôlée ou non (checked/unchecked)
- Une méthode peut contrôler elle même les exceptions :

```
public void lire() {
        try{
                return (new FileReader(monFichier));
        } catch (FileNotFoundException e){  //traitement erreur
        }
}
```

- … ou non

```
public void lire() throws FileNotFoundException{
        … // autre code
        return (new FileReader(monFichier));
}
```

A savoir

- Les blocs try-catch peuvent avoir multiples blocs catch
  - Mais il n'y a qu'un bloc try, qui contient un code commun
  - Si blocs catch multiples : exceptions en hiérarchie inversée
    - Sous-classes avant superclasses : NullPointerException avant RuntimeException
- Les blocs try peuvent avoir un bloc finally après 0, 1, 2... blocs catch
  - Il s'exécute obligatoirement
  - Il permet de bien clôturer des processus ouverts (e.g. un fichier ouvert)
- Deux instructions utiles :
  - System.err.println(String) : une méthode qui permet d'afficher des textes lors du lever d'une exception
  - printStackTrace() : méthode applicable à tout objet qui implémente l'interface Throwable; elle trace la source de l'erreur ou exception

Un exemple : index out of Bounds

Regardons ce code

MonException.java

```java

public class MonException {
    public static void main(String[] args) {
        MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
        int taille = 3;
        int[] monTableau = new int[taille];
        int monEntree = 10;
        monTableau[0] = monEntree;
        System.out.println("La premiere composante du tableau est " + monTableau[0]);
        objet.doubler(monEntree);
        monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
        System.out.println("La deuxieme composante du tableau est " + monTableau[1]);

    }

    public int augmenter (int index) {
        System.out.println("La valeur initiale de l'index est " + index);
        index = index + 1;
        System.out.println("L'index a ete augmente a " + index);
        return index;
    }

    public int doubler (int valeur) {
        System.out.println("Nous allons doubler la valeur " +valeur);
        valeur = valeur * valeur;
        System.out.println("Valeur doublee " + valeur);
        return valeur;
    }
}

```

Lors d'une execution normale

Problems Javadoc Declaration Console

<terminated> MonException [Java Application] C:\Program Files\Java

```
La premiere composante du tableau est 10
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
La deuxieme composante du tableau est 100
```

Avec une erreur

ErrorProgram/src/MonException.java

```
1
2  public class MonException {
3      public static void main(String[] args) {
4          MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
5          int taille = 1;
6          int[] monTableau = new int[taille];
7          int monEntree = 10;
8          monTableau[0] = monEntree;
9          System.out.println("La premiere composante du tableau est " + monTableau[0]);
10         objet.doubler(monEntree);
11         monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
12         System.out.println("La deuxieme composante du tableau est " + monTableau[1]);
13
14     }
15
```

taille changée

Problems Javadoc Declaration Console

<terminated> MonException [Java Application] C:\Program Files\Java\jre1.8.0_

```
La premiere composante du tableau est 10
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
Exception in thread "main" L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
java.lang.ArrayIndexOutOfBoundsException: 1
        at MonException.main(MonException.java:11)
```

Lever l'exception

```
1
2  public class MonException {
3      public static void main(String[] args) {
4          MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
5          int taille = 1;
6          int[] monTableau = new int[taille];
7          int monEntree = 10;
8          monTableau[0] = monEntree;
9          System.out.println("La premiere composante du tableau est " + monTableau[0]);
10         objet.doubler(monEntree);
11         try {
12             monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
13             System.out.println("La deuxieme composante du tableau est " + monTableau[1]);
14         }
15         catch(IndexOutOfBoundsException e) {
16             System.out.println("Taille du tableau depassee !");
17         }
18     }
19
20     public int augmenter (int index) {
21         System.out.println("La valeur initiale de l'index est " + index);
22         index = index + 1:
```

Problems Javadoc Declaration Console

```
<terminated> MonException [Java Application] C:\Program Files\Jav
La premiere composante du tableau est 10
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
Taille du tableau depassee !
```

try-catch

cherche une exception du type IndexOutOfBounds

exception levée

Tracer la source de l'erreur

```java
public class MonException {
    public static void main(String[] args) {
        MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
        int taille = 1;
        int[] monTableau = new int[taille];
        int monEntree = 10;
        monTableau[0] = monEntree;
        System.out.println("La premiere composante du tableau est " + monTableau[0]);
        objet.doubler(monEntree);
        try {
            monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
            System.out.println("La deuxieme composante du tableau est " + monTableau[1]);
        }
        catch(IndexOutOfBoundsException e) {
            System.err.println("Taille du tableau depassee !");
            e.printStackTrace();
        }
        finally {
            System.out.println("Dans le bloc finally");
        }
    }
```

Problems Javadoc Declaration Console

<terminated> MonException [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Jan 17, 2018, 1:43:26 PM)

```
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
Taille du tableau depassee !
java.lang.ArrayIndexOutOfBoundsException: 1
        at MonException.main(MonException.java:12)
Dans le bloc finally
```

System.out.println remplacé par System.err.println

tracer l'erreur

Tracer la source de l'erreur

```java
public class MonException {
    public static void main(String[] args) {
        MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
        int taille = 1;
        int[] monTableau = new int[taille];
        int monEntree = 10;
        monTableau[0] = monEntree;
        System.out.println("La premiere composante du tableau est " + monTableau[0]);
        objet.doubler(monEntree);
        try {
            monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
            System.out.println("La deuxieme composante du tableau est " + monTableau[1]);
        }
        catch(IndexOutOfBoundsException e) {
            System.err.println("Taille du tableau depassee !");
            e.printStackTrace();
        }
        finally {
            System.out.println("Dans le bloc finally");
        }
    }
```

Problems Javadoc Declaration Console

```
<terminated> MonException [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Jan 17, 2018, 1:43:26 PM)
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
Taille du tableau depassee !
java.lang.ArrayIndexOutOfBoundsException: 1
        at MonException.main(MonException.java:12)
Dans le bloc finally
```

Un bloc finally

Nouvelles exceptions & bonnes pratiques

Créer une nouvelle exception

- Nous pouvons créer de nouvelles exceptions en Java

BP1 : Toujours d'abord utiliser les exceptions qui existent déjà en Java !

- Si contrôlée, l'exception hérite impérativement de la classe Exception
- Si non-contrôlées, l'exception hérite de la classe RuntimeException
- La nouvelle exception formera une nouvelle classe, avec des attributs et des méthodes.
  - N'oubliez pas d'utiliser l'héritage d'Exception/RuntimeException !

Exemple : Etudiant non-trouvé

```
public class EtudiantNonTrouve extends RuntimeException(){
        // constructeur personalisé
        public EtudiantNonTrouve(String m){
                super(message); // nous utilisons le constructeur d'Exception
        }
}
```

```
public class classePrincipale{
        public static void main(String[] args) {
                Etudiant[] mesEtudiants;
                for (int i = 0; i < mesEtudiants.length(); i++){
                        if (mesEtudiants[i].getNom().equals("Dupont")){
                                ...
                        }
                        else
                                throw new EtudiantNonTrouve("Aucun Dupont dans la classe !");
                }
        } }
```

D'autres bonnes pratiques

- Important de capturer toute exception impactant notre code

BP2 : Utiliser les sous-classes spécifiques des exceptions, plutôt que les superclasses (Exception/RuntimeException)

- Les blocs try-catch-finally ont des structures spéciales

BP3 : Ne jamais utiliser les blocs catch{} comme branche alternative d'execution (c'est pourquoi on a if-then-else !)

BP4 : Ne jamais utiliser `return` lorsqu'une exception est levée