Principe : montrer qu'en imposant quelques contraintes d'écriture de code (nommage notamment), on permet la prise en compte de classes inconnues initialement. L'externalisation du paramétrage de la création de jeux prend donc tout son sens.
Dans la suite, lorsqu'il sera indiqué "on obtient" pour les objets String cela signifie que le contenu de la chaîne provient de la lecture du fichier XML, il s'agit des "`#PCDATA`" lues. Ces fichiers XML sont conformes à la DTD fixée et connue à l'avance, on peut donc se baser sur la grammaire qu'elle définit pour exploiter parcourir le fichier XML. Voici la DTD utilisée pour illustrer ce texte :

```
<!ELEMENT game (...,entity*,...)>
<!ELEMENT entity (className,number,param*,active?)
<!ELEMENT className (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT param (#PCDATA)>
<!ELEMENT active (ability*)>
<!ELEMENT ability (#PCDATA)>
```

Ainsi si `root` représente la racine (`game`) d'un document XML conforme à cette DTD, `root.getChildren("entity")` permet de récupérer la liste des fils éléments ayant pour balise "`entity`". Cette liste peut être parcourue, et pour un de ses éléments `entityElement`, `entityElement.getChildText("className")` permet d'accéder à l'information textuelle (`String`) représentant le nom de la classe de cette entité.
On obtient ainsi cette information. On procède de manière similaire pour parcourir l'ensemble du document.

Création d'instances

Hypothèses : il existe un constructeur dont la signature est connue (ici il prend une chaîne de caractères en paramètre) et une méthode `init(String...)`[1] qui réalise l'initialisation de l'objet. Il faut dans cette méthode convertir les chaînes de caractères paramètres vers les types voulus.
On suppose dans la suite que les capacités, du type `Ability`, ont un constructeur sans paramètre.

Le code (qui n'est donné ici que pour comprendre la suite, il ne correspond a priori pas au code du projet) :

```
public abstract class Entity {
   protected String name;
   protected List<Ability> abilities;
   public Entity(String name) {
      this.name = name;
   }
   /** this method is used to initialize an entity object from String values
   * @param args the String values used to initialize the entity object
   * (ie.  to initialize fields) */
   public abstract void init(String...  args);
   public void addAbility(Ability ability) {
      this.abilities.add(ability);
   }
}
```

L'exploitation pour la création avec *obtention* des données depuis fichier XML (par exemple) :

```
         // on obtient une chaîne représentant la classe de Entity à créer
         //  c'est le nom complet :  "truc.machin.MyEntityClass"
String className = ...
         // on obtient un tableau des String représentant des
         //  paramètres de création de l'Entity (ils sont obtenus un par un
         //  et rangés dans le tableau)
String[] args = ...
         // récupère l'objet représentant la classe de l'entity
Class entityClass = Class.forName(className);
         // on construit arbitrairement le nom de l'instance à l'aide du nom
         //  de la classe
```

[1] Rappelons que la syntaxe "`T... t`" signifie "*un nombre quelconque de paramètres de type `T`*", syntaxe disponible depuis java 1.5 et qui n'est possible qu'en fin de signature de méthode. Dans le corps de la méthode les paramètres sont traités comme si l'on n'avait qu'un paramètre `t` de type `T[]`, ils sont donc identifiés par `t[i]`.

```
String name = className+(compteur++);
         // on récupère le constructeur qui prend un String en paramètre
Constructor constructor = entityClass.getConstructor(String.class);
Entity entity = constructor.newInstance(name);
         // appel de la méthode init(String...)  qui initialise les attributs
         //  à partir d'objets String
entity.init(args);
         // on obtient le nom d'une capacite de l'entite
String abilityClassName = ...;
         // on récupère l'objet classe
Class abilityClass = Class.forName(abilityClassName);
         // creation de l'instance avec constructeur sans paramètre
Ability ability = abilityClass.newInstance();
         // on ajoute la capacité à l'entité
entity.addAbility(ability);
...
```

Un exemple de classe de "Entity" exploitable et qui peut être créée après l'écriture des codes précédents :

```
package projet.entity;
public class MyEntity extends Entity {
   protected int speed;
   protected int age;

   public MyEntity(String name) {
      this(name,0,1);
   }
   public MyEntity(String name, int speed, int age) {
      super(name);
      this.init(speed, age);
   }
   private void init(int speed, int age) {
      this.speed = speed;
      this.age = age;
   }
   /**
   * @param args [0] = speed of this entity, [1] = age of this entity
   */
   public void init(String...  args) {
      this.init(Integer.parseInt(args[0]),Integer.parseInt(args[1]));
   }
   public String toString() {
      return "Entity "+this.name;
   }
   (...)
}
```

On peut alors retrouver dans un fichier XML, qui serait conforme à la DTD fournie, quelque chose comme ceci :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE simulation SYSTEM "game.dtd">
<game>
(...)
   <entity>
      <className>projet.entity.MyEntity</className>
      <number>4</number>
      <param>2</param>   <!-- speed -->
      <param>5</param>   <!-- age -->
      <active>
         <ability>projet.ability.MoveCommand</ability>
      </active>
   </entity>
(...)
</simulation>
```

qui, exploité avec un code proche du précédent, permettrait la création de 4 entités correspondant à cette classe dans la simulation décrite par ce fichier.

Ceci n'est qu'un exemple donné à titre d'illustration.