Version XML

Fichier de configiration

On a dans cet exemple deux BEans, Utilisateur et Compte, dans le deuxiemens BEan on remarque qu'il depende du premier Bean

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<grains>
    <grain id="1" classe="Utilisateur" >
        <propriete nom="password" value="ilass"/>
        <propriete nom="nom" value="hassan1"/>
    </grain>
    <grain id="2" classe="Utilisateur"  >
        <propriete nom="password" value="hassan"/>
        <propriete nom="nom" value="hassan1"/>

    </grain>
    <grain id="3" classe="compte">
        <constructeur nom="utilisateur" ref="1"/>
    </grain>
</grains>
```

Entities

Bean

```java
package Traitement;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import java.util.List;

@XmlAccessorType(XmlAccessType.FIELD)
public class Grain {
    @XmlAttribute
    String id ;
    @XmlAttribute
    String  classe;

    List<Proprietes> propriete;
    Constructeur constructeur ;

    public Grain(String id, String classe, List<Proprietes> proprietes) {
        this.id = id;
        this.classe = classe;
        this.propriete = proprietes;
    }

    public Grain(String id, String classe, Constructeur contructeur) {
        this.id = id;
```

```
        this.classe = classe;
        this.constructeur = contructeur;
    }

    public Grain() {
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getClasse() {
        return classe;
    }

    public void setClasse(String classe) {
        this.classe = classe;
    }

    public List<Proprietes> getPropriete() {
        return propriete;
    }

    public void setPropriete(List<Proprietes> propriete) {
        this.propriete = propriete;
    }

    public Constructeur getContructeur() {
        return constructeur;
    }

    public void setContructeur(Constructeur contructeur) {
        this.constructeur = contructeur;
    }

    @Override
    public String toString() {
        return "Grain{" +
                "id='" + id + '\'' +
                ", classe='" + classe + '\'' +
                ", propriete=" + propriete +
                '}';
    }
}
```

Beans

Cette classe represente l'element racine du fichier de configuration

```
package Traitement;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
```

```java
import javax.xml.bind.annotation.XmlRootElement;
import java.util.List;

@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Grains {

    List<Grain> grain;

    public Grains(){

  }
    public Grains(List<Grain> grainlist) {
        this.grain = grainlist;
    }

    public List<Grain> getGrainlist() {
        return grain;
    }

    public void setGrainlist(List<Grain> grainlist) {
        this.grain = grainlist;
    }

    @Override
    public String toString() {
        return "Grains{" +
                "grain=" + grain +
                '}';
    }
}
```

information sur l'objet :

```java
package Traitement;

public class InfoObjet {
    String Classe;
    String Id  ;

    public InfoObjet(String classe, String id) {
        Classe = classe;
        Id = id;
    }

    public String getClasse() {
        return Classe;
    }

    public void setClasse(String classe) {
        Classe = classe;
    }

    public String getId() {
        return Id;
    }

    public void setId(String id) {
        Id = id;
    }
```

```
    @Override
    public String toString() {
        return "InfoObjet{" +
                "Classe='" + Classe + '\'' +
                ", Id='" + Id + '\'' +
                '}';
    }
}
```

injection des depe par les setters :

```
HashMap<InfoObjet, Object> objects = new HashMap<>();
ApplicationMain app = new ApplicationMain("Config.xml");
Grains grs = (Grains) app.getObjects();
Boolean created;

for (Grain grain : grs.getGrainlist()) {
    created = false;
    Class gr = Class.forName(grain.getClasse());

    List<Proprietes> p1 = grain.getPropriete();
    Constructeur ctr = grain.getContructeur();

    if(p1!=null) {

        Object a =  gr.newInstance();
        objects.put(new InfoObjet(grain.getClasse(),grain.getId()),a);
        for(Proprietes p:p1)
        if (p.getValue()==null) {
            for (InfoObjet j : (Set<InfoObjet>) objects.keySet()) {
                if (p.getRef().equals(j.getId())) {
                    Class inj = Class.forName(j.getClasse());
                    Method mth = gr.getMethod("set" + p.getNom(), inj);
                    mth.invoke(a, objects.get(j));
                }
            }
            created=true;
        }else{
            Method mth = gr.getMethod("set" + p.getNom(),String.class);
            mth.invoke(a, p.getValue());
                created=true;
        }
    }
```

injection des depe par le cnostructeur :

```
if (ctr != null) {
    InfoObjet tmpKey = null;
    Object tmpValue = null;
    for (InfoObjet j : (Set<InfoObjet>) objects.keySet()) {
        if (ctr.getRef().equals(j.getId())) {
            tmpValue = objects.get(j);
            tmpKey = j;
        }
```

```
    }
    System.out.println(tmpKey.getClasse());
    Class inj = Class.forName(tmpKey.getClasse());
    Class<?> cl = Class.forName(grain.getClasse());
    Constructor<?> cons = cl.getConstructor(inj);
    Object o = cons.newInstance(tmpValue);
    objects.put(new InfoObjet(grain.getClasse(), grain.getId()), o);
    created = true;
}
```

injection par les proprietes :

```
if (p1 != null) {
    Object a = gr.newInstance();
    objects.put(new InfoObjet(grain.getClasse(), grain.getId()), a);
    for (Proprietes p : p1)
        if (p.getValue() == null) {
            for (InfoObjet j : (Set<InfoObjet>) objects.keySet()) {
                if (p.getRef().equals(j.getId())) {
                    Class inj = Class.forName(j.getClasse());
                    Class<?> c = a.getClass();
                    Field f = c.getDeclaredField(p.getNom());
                    f.setAccessible(true);
                    f.set(a, objects.get(j));
                }
            }
        } else {
            Class<?> c = a.getClass();
            Field f = c.getDeclaredField(p.getNom());
            f.setAccessible(true);
            f.set(a, p.getValue());
        }
    created = true;
}
```

La recuperations des objets :

```
import Traitement.Grains;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import java.io.File;

public class ApplicationMain {

    private  String FileName ;
    public ApplicationMain(String FileName){
        this.FileName = FileName ;
    }


    public Object getObjects () throws JAXBException{
        JAXBContext context = JAXBContext.newInstance(Grains.class);
        Unmarshaller unmarshaller = context.createUnmarshaller();
        Grains grains =(Grains) unmarshaller.unmarshal(new File(FileName));
        return grains;
```

```
    }
}
```

Les annotations :

Annotation pour les attributes :

```
package Annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)

public @interface Attribut {

}
```

Component :

```
package Annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Composant {
}
```

Annotation pour les constructeurs :

```
package Annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;


@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.CONSTRUCTOR)

public @interface Constructeur {
}
```

Annotation pour les seters :

```

package Annotation;

import java.lang.annotation.ElementType;
```

```java
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)

public @interface Set {
}
```

Traitement :

```java
int i = 0;
    //Hashmap to store beans

    HashMap<String, Object> store = new HashMap<>();
    HashMap<Class, Object> fieldStore = new HashMap<>();
    for (Class c : getClasses("Annotation.ToScan", false)) {
        // Field classMemberField =
valids.class.getDeclaredField("classMember");
        //prepare the store
        for (Annotation ann : c.getAnnotations()) {
            if (ann instanceof Composant) {   // make sure it's your
annotation
                String s[] = c.getName().split("\\.");
                if (s.length != 0)
                    store.put(s[s.length - 1], c.newInstance());

            }
        }
        Field[] fields = c.getDeclaredFields();
        for (int j = 0; j < fields.length; j++) {
            Annotation[] annotations = fields[j].getDeclaredAnnotations();
            //checking if the field is annotated

            if (annotations.length != 0) {

                fieldStore.put(fields[j].getType(), fields[j]);
            }
        }
        System.out.println(fieldStore);
        int k = 0;

        for (int j = 0; j < fieldStore.size(); j++) {

            System.out.println("size" + fieldStore.size());
        }

        for (Map.Entry fi : fieldStore.entrySet()) {
            //System.out.println(fi.getName());
            for (String nom : store.keySet()) {
                Class cls = Class.forName("Annotation.ToScan." + nom);
```

```
                Class cs = (Class) fi.getKey();
                if (cs.isAssignableFrom(cls)) {

                    Field f = (Field) fi.getValue();

                    Object ob = f.getDeclaringClass().newInstance();
                    f.setAccessible(true);
                    main m = new main();
                    f.set(ob, m);
                    valids v = (valids) ob;
                    ((valids) ob).getNumero().test();
                    store.put(((Class<?>) fi.getKey()).getName(), ob);
                }
            }
        }

        //test
        for (Map.Entry fi : store.entrySet()) {
            fi.getValue().toString();
        }

    }

    i++;
}

public static Class<?>[] getClasses(String packageName, boolean recursive)
throws ClassNotFoundException, IOException {
    ComponentContainer componentConatiner =
ComponentContainer.getInstance();
    ClassHunter classHunter = componentConatiner.getClassHunter();
    String packageRelPath = packageName.replace(".", "/");
    SearchConfig config = SearchConfig.forResources(
            packageRelPath
    );
    if (!recursive) {
        config.findInChildren();
    }

    try (ClassHunter.SearchResult result = classHunter.findBy(config)) {
        Collection<Class<?>> classes = result.getClasses();
        return classes.toArray(new Class[classes.size()]);
    }
}
```