Pattern

Soit une interface Job representant un job, un stage. Les jobs ont a minima un titre et un identifiant

Job

```
public interface Job {

	Long getId();
	String getTitle();
}
```

Les Stages

Les Stages sont des types de Job

```
public class Stage implements Job{

	private String title;
	private Long id;
	
	public Stage()
	{
		
	}
	public String getTitle() {
		return title;
	}

	public Long getId() {
		return id;
	}

	@Override
```

```
    public String toString() {
        return "Stage [title=" + title + ", id=" + id + "]";
    }
```

Pattern de builder

Le pattern de builder permet de construire des objets sans pouvoir les modifier (object immuable):

```
private Stage(Builder builder) {
    this.title = builder.title;
    this.id = builder.id;
}
/**
 * Creates builder to build {@link Stage}.
 * @return created builder
 */
public static Builder builder() {
    return new Builder();
}
/**
 * Builder to build {@link Stage}.
 */
public static final class Builder {
    private String title;
    private Long id;

    private Builder() {
    }

    public Builder withTitle(String title) {
        this.title = title;
        return this;
    }

    public Builder withId(Long id) {
        this.id = id;
        return this;
    }
```

```
    public Stage build() {
        return new Stage(this);
    }
}
```

Pattern de Singleton

La librairie Kryo permet de sauvegarder des grappes d'objet sur un fichier (et de les relire).

La dépendance est la suivante:

```
 <dependency>
   <groupId>com.esotericsoftware</groupId>
   <artifactId>kryo</artifactId>
   <version>5.0.0-RC4</version>
</dependency>
```

Le pattern de singleton est implémenté via un enum:

```
public enum KrioSingleton {

    INSTANCE;
    private Kryo kryo;

    private KrioSingleton()
    {
        this.kryo = new Kryo();
        kryo.register(java.util.HashMap.class);
        kryo.register(JobRepository.class);
        kryo.register(Stage.class);

    }
    public JobRepository read(String filename) throws FileNotFoundException
    {
        Input input = new Input(new FileInputStream(filename));
        JobRepository jobRepository = kryo.readObject(input, JobRepository.class);
        input.close();
        return jobRepository;
    }
```

```
public void save(String filename,JobRepository repository) throws FileNotFoundException
{
    Output output = new Output(new FileOutputStream(filename));
        kryo.writeObject(output, repository);
        output.flush();
        output.close();
}

}
```

Pattern de DAO.

Un pattern de DAO est capable de lire et d'ecrire un objet depuis une base de donnée, fichier ... Ici on fait une implémentation avec Kryo:

```
public class JobRepository {


Map<Long,Job> data=new HashMap<Long,Job>();
static String defaultFileName="jobstore.bin";
public JobRepository()
{

}
public static JobRepository getJobRepository()
{
    try {
        return KrioSingleton.INSTANCE.read(defaultFileName);
    } catch (FileNotFoundException e) {
        return new JobRepository();
    }
}
public Long nextId()
{
    return new Long(data.size()+1);

}
public Job readJob(Long id)
{
    return data.get(id);
}
public void saveJob(Job job)
```

```
    {
        this.data.put(job.getId(), job);
        try {
            KrioSingleton.INSTANCE.save(defaultFileName, this);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
    @Override
    public String toString() {
        return "JobRepository [data=" + data + "]";
    }

}
```

Revision #2
Created 4 November 2019 15:46:40 by Admin
Updated 4 November 2019 15:57:44 by Admin