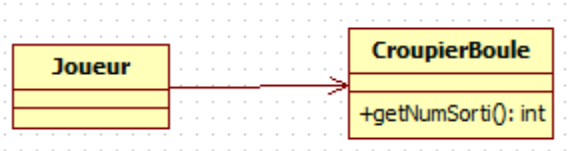
TP Mock

Pour faire ce TP il faut récupérer `mockito-all-x.x.x.jar` à partir du site `mockito.org`.
On veut modéliser un jeu de casino : le jeu de la boule.

Règle du jeu :
Le jeu de la boule est un jeu de casino simplifié du jeu de la roulette. Il utilise les chiffres de 1 à 9. Le joueur qui joue fait un pari en misant une somme.
Il est possible de miser sur rouge, noir, manque, passe, pair ou impair.
Les chiffres 1, 3, 7, 9 sont impairs. Les chiffres 2, 4, 6, 8 sont pairs.
Les chiffres 1, 3, 6, 8 sont noirs. Les chiffres 2, 4, 7, 9 sont rouges.
Les chiffres 1, 2, 3, 4 sont "manque" ("on a manqué de dépasser 5"). Les chiffres 6, 7, 8, 9 sont "passe" ("on a dépassé 5").
Ces chances sont des chances simples : si la chance simple misée sort, le joueur gagne une fois la mise (qui lui est restituée), sinon la mise est perdue.
Le chiffre 5 n'est ni pair, ni impair, ni manque, ni passe, ni rouge, ni noir. Si le 5 sort, la mise jouée sur une chance simple est perdue.
Le joueur peut miser sur un numéro. Si celui-ci sort, le joueur gagne 7 fois la mise qui lui est restituée sinon la mise est perdue.
Un joueur peut évidemment miser sur plusieurs cases (et même sur rouge et noir !). Il ne peut miser que des quantités entières (des jetons de valeur entière en euro).

Pour modéliser ce jeu on utilise aux moins deux classes : la classe `Joueur` qui modélise un ... joueur et la `CroupierBoule` qui modélise le gestionnaire du jeu de la boule : le croupier. La classe `CroupierBoule` possède la méthode
`public int getNumSorti()` qui retourne le numéro sorti. On a donc :



Développement de la classe à isoler (la classe `Joueur`) et de son mock associé

1°) Faut-il développer ce logiciel avec seulement ces deux classes ? Penser qu'on nous demandera sûrement un jour de faire de même avec le jeu de la roulette. Indiquer alors la (es) partie(s) logicielle(s) manquante(s). Quel est le principe de génie logiciel orienté objet utilisé ici ?

une réponse :
Il ne faut pas faire le logiciel avec seulement des classes : il faut abstraire les fonctionnalités et les placer dans des interfaces Java. Les fonctionnalités du casino sont mises dans l'interface `CasinoInterface` suivante :

```
public interface CasinoInterface {
	// Ce que doit rendre comme service un casino
	// (indiquer le numéro sorti, indiquer les gains et perte, etc.)
}
```

Cette interface sera implémentée par les classes `CroupierBoule`, `CroupierRoulette`, etc.
Le principe de génie logiciel orienté objet que nous illustrons ici est l'injection de dépendance.

2°) Un joueur est lié au casino et c'est le casino qui lui indique combien il a gagné ou perdu. Il peut savoir quel numéro est sorti en le demandant au casino (ou au représentant du casino). La classe `Joueur` possède donc les méthodes :

```
public int aGagneOuPerdu() {
	// demander au casino combien le joueur a gagné ou perdu et retourne ce gain
	// ou perte
}

public int getNumeroSorti() {
	// demander au casino le numéro sorti
}
```

Ecrire la classe `Joueur`. C'est la classe à tester et on veut l'isoler de la classe qui modélise le casino (son mock).
Coder ces deux méthodes de sorte qu'elles demandent leur résultat au représentant du casino. De ce fait toutes les décisions, le numéro sorti, les gains ou pertes des joueurs, plus tard les mises faites par le joueur sur les diverses cases, sont prises par le représentant du casino : le joueur ne fait que les demandes.
Finalement utilise-t-on la classe `CroupierBoule` ?

une réponse :
Voici la classe `Joueur` :

```
package jeu.boule;


public class Joueur {
	private CasinoInterface leCasino;

	public void setLeCasino(CasinoInterface leCasino) {
		this.leCasino = leCasino;
	}
```

```
	public CasinoInterface getLeCasino() {
		return leCasino;
	}

	public int aGagneOuPerdu() {
		return leCasino.gainOuPerte();
	}

	public int getNumeroSorti() {
		return leCasino.getNumSorti();
	}
}
```

Comme on peut le voir on n'utilise pas la classe `CroupierBoule` mais son abstraction l'interface `CasinoInterface`.

3°) La classe de test qui teste la classe `Joueur` doit être :

```
package test;

import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

import org.junit.Before;
import org.junit.Test;

import jeu.boule.CasinoInterface;
import jeu.boule.Joueur;

public class JoueurTestJUnit4 {
	private Joueur leJoueur;
	private CasinoInterface laDoublure;

	@Before
	public void setUp() {
		leJoueur = new Joueur();
		laDoublure = mock(CasinoInterface.class);
		leJoueur.setLeCasino(laDoublure);
	}

	@Test
	public void testGagne() {
	// On suppose que c'est le joueur n'a joué que sur le 8 avec
	// 3 jetons et que le 8 sort
		System.out.println("Le joueur n'a joué que sur le 8 avec 3 jetons et le
8 est sorti");
		when(laDoublure.getNumSorti()).thenReturn(8);
		when(laDoublure.gainOuPerte()).thenReturn(21);
		int gagneOuPerdu = leJoueur.aGagneOuPerdu();
		System.out.println("Le joueur a gagné : " + gagneOuPerdu + " parce que
le numéro sorti est : " + leJoueur.getNumeroSorti());
		assertEquals(21, gagneOuPerdu);

		System.out.println("fin de testGagne");
	}
```

```
	@Test
	public void testPerdu() {
	// On suppose que c'est le joueur n'a joué que sur le 8 avec
	// 3 jetons et que le 9 est sorti
		System.out.println("Le joueur n'a joué que sur le 8 avec 3 jetons et le
9 est sorti");
		when(laDoublure.getNumSorti()).thenReturn(9);
		when(laDoublure.gainOuPerte()).thenReturn(-3);
		int gagneOuPerdu = leJoueur.aGagneOuPerdu();
		System.out.println("Le joueur a gagné : " + gagneOuPerdu + " parce que
le numéro sorti est : " + leJoueur.getNumeroSorti());
		assertEquals(-3, gagneOuPerdu);
		System.out.println("fin de testPerdu");
	}
}
```

Ecrire cette classe de test. Lancer le test.
une réponse :
La classe qui lance le test est celle-ci-dessus et elle est entièrement donnée. On a construit un mock qui concrétise une abstraction du casino est qui fonctionne correctement en supposant les mises du Joueur et le numéro sorti.

4°) Concevoir d'autres jeux de test.

une réponse :
Il suffit de suivre le plan de la classe précédente avec d'autres mises et d'autres numéros sortis par le jeu de la boule.

Remarque :
Il faut être conscient que nous n'avons pas fait … grand-chose ! On a testé une classe (`Joueur`) qui utilise essentiellement une classe externe qui est, dans l'exercice, représentée par un mock et qui donc renvoie des résultats justes (ils ont été codés dans le mock). Mais :
1°) On veut illustrer les mocks dans cet exercice
2°) Le codage de la classe externe est proposé en bonus dans la suite de l'exercice.

une réponse :
Ces remarques sont peut-être décevantes mais elles sont exactes. D'ailleurs pour s'en convaincre voir le code complet de la classe `CroupierBoule` en bonus ci-dessus : il n'est pas complètement trivial.

Bonus (sans les mocks) : implémentation de la classe `CroupierBoule`

5°) On veut enrichir le représentant du casino des méthodes :

```
	public void addMiseSimple(String st, Integer i);
	public void addMiseNumero(Integer numero, Integer quantite);
	public void indiqueNumeroSorti ();
```

Les méthodes `addXXX()` permettent à un joueur de miser. Elles seront appelées par les méthodes `public void depotMiseSimple(String nomMise, int quantite)` et `public void depotMiseNumero(int numero, int quantite)`
du joueur.

La méthode `indiqueNumeroSorti()` initialise la donnée membre indiquant le numéro "choisi par la boule" (sorti).

Ecrire la classe `CroupierBoule`. Cette classe construit le résultat du tirage du numéro, stocke les mises sur le tapis d'un seul joueur (elle devra être plus fournie pour plusieurs joueurs plus tard), calcule et retourne les gains et pertes du joueur par sa méthode
`public int gainOuPerte()`.
Cette classe peut commencer par :

```
public class CroupierBoule implements CasinoInterface {
  private HashMap<String, Integer> lesMisesSimples = new HashMap<String, Integer>();
  private HashMap<Integer, Integer> lesNumeros = new HashMap<Integer, Integer>();
  private int resultat;
```

S'en servir dans un test (sans les mocks) pour vérifier qu'un joueur ayant misé :
3 sur le numéro 8
3 sur le numéro 9
5 sur le noir
2 sur pair
15 sur passe
a gagné 40 lorsque le numéro 8 est sorti.
Si le numéro 5 était sorti, ce joueur aurait perdu 28.

une réponse :
Le code de la classe `CroupierBoule` est :

```
import java.util.HashMap;
import java.util.Set;

public class CroupierBoule implements CasinoInterface {
  private HashMap<String, Integer> lesMisesSimples = new HashMap<String, Integer>();
  private HashMap<Integer, Integer> lesNumeros = new HashMap<Integer, Integer>();
  private int resultat;

  public CroupierBoule() {
    super();
  }

  public int getNumSorti() {
    return resultat;
  }

  public void construitResultat() {
    // resultat = (int)(Math.random() * 9) + 1;
    resultat =  8;
  }

  public void addMiseSimple(String st, Integer quantite) {
    lesMisesSimples.put(st, quantite);
  }

  public void addMiseNumero(Integer numero, Integer quantite){
    lesNumeros.put(numero, quantite);
  }

  public int gainOuPerte() {
    // Set misesSimplesJouees = lesMisesSimples.keySet();
    System.out.println("debut de gainOuPerte() du jeu de la boule");
    int gainOuPerteFinal = 0;
```

```
    int leResultat = getNumSorti();
    boolean pasDeGain = true;

    // Calcul des gains et pertes sur les numeros pleins
    Set<Integer> LesNumerosJoues = lesNumeros.keySet();
    for (Integer unNumeroJoue : LesNumerosJoues) {
      Integer quantiteSurCeNumero = lesNumeros.get(unNumeroJoue);
      pasDeGain = true;
      int numeroEtudie = unNumeroJoue.intValue();
      if (numeroEtudie == leResultat) {
        System.out.println("un gain sur le numero " + leResultat);
        gainOuPerteFinal += 7 * quantiteSurCeNumero;
        pasDeGain = false;
      } else if (quantiteSurCeNumero != 0) {
        System.out.println("perte de " +  quantiteSurCeNumero + " sur le numero " +
numeroEtudie);
        gainOuPerteFinal -= quantiteSurCeNumero;
      }
    }

    // Calcul des gains et pertes sur les mises simples
    // D'abord les initialisations
    // Pour la couleur
    String laCouleurSortie = "";
    if (leResultat == 1 || leResultat == 3 || leResultat == 6 || leResultat == 8) {
      laCouleurSortie = Constantes.NOIR;
    } else if (leResultat == 2 || leResultat == 4 || leResultat == 7 || leResultat ==
9) {
      laCouleurSortie = Constantes.ROUGE;
    }
    // Pour passe ou manque
    String passeOuManqueSorti = "";
    if (leResultat > 5) {
      passeOuManqueSorti = Constantes.PASSE;
    } else if (leResultat < 5) {
      passeOuManqueSorti = Constantes.MANQUE;
    }
    // fin des initialisations

    // Calcul proprement dit des gains et pertes sur les mises simples
    Set<String> lesChancesSimplesJouees = lesMisesSimples.keySet();
    for (String uneMiseChanceSimpleJouee : lesChancesSimplesJouees) {
      Integer quantiteSurCetteChanceSimple =
lesMisesSimples.get(uneMiseChanceSimpleJouee);
      if (leResultat == 5) {
        gainOuPerteFinal -= quantiteSurCetteChanceSimple;
        System.out.println("Le 5 est sorti donc perte sur " +
uneMiseChanceSimpleJouee);
        continue;
      }
      // D'abord les parites
      if ((uneMiseChanceSimpleJouee.equals(Constantes.PAIR)) && (leResultat % 2 == 0))
{
        System.out.println("même parité");
        gainOuPerteFinal += quantiteSurCetteChanceSimple;
        pasDeGain = false;
      }
      if ((uneMiseChanceSimpleJouee.equals(Constantes.IMPAIR)) && (leResultat % 2 ==
1)) {
        System.out.println("même parité");
        gainOuPerteFinal += quantiteSurCetteChanceSimple;
        pasDeGain = false;
      }

      // ensuite les couleurs
      if ((uneMiseChanceSimpleJouee.equals(Constantes.NOIR)) &&
(laCouleurSortie.equals(Constantes.NOIR))) {
```

```
        System.out.println("même couleur noire");
        gainOuPerteFinal += quantiteSurCetteChanceSimple;
        pasDeGain = false;
      }
      if ((uneMiseChanceSimpleJouee.equals(Constantes.ROUGE)) &&
(laCouleurSortie.equals(Constantes.ROUGE))) {
        System.out.println("même couleur rouge");
        gainOuPerteFinal += quantiteSurCetteChanceSimple;
        pasDeGain = false;
      }

      // Enfin passe ou manque
      if ((uneMiseChanceSimpleJouee.equals(Constantes.PASSE)) &&
(passeOuManqueSorti.equals(Constantes.PASSE))) {
        System.out.println("même passe");
        gainOuPerteFinal += quantiteSurCetteChanceSimple;
        pasDeGain = false;
      }
      if ((uneMiseChanceSimpleJouee == Constantes.MANQUE) &&
(passeOuManqueSorti.equals(Constantes.MANQUE))) {
        System.out.println("même manque");
        gainOuPerteFinal += quantiteSurCetteChanceSimple;
        pasDeGain = false;
      }
      if (pasDeGain) {
        gainOuPerteFinal -= quantiteSurCetteChanceSimple;
      }
      System.out.println(" qui amène un gain ou une perte de " + gainOuPerteFinal);
    }
    System.out.println("FIN de gainOuPerte() de la Boule");
    return gainOuPerteFinal;
  }
}
```

Avec l'interface :

```
public interface Constantes {
      public static String NOIR = "noir";
      public static String ROUGE = "rouge";
      public static String PAIR = "pair";
      public static String IMPAIR = "impair";
      public static String PASSE = "passe";
      public static String MANQUE = "manque";
}
```

On comprend désormais pourquoi avoir un mock qui donnait immédiatement le résultat en fonction des mises et du numéro sorti était bien pratique et dispense de vérifier que l'algorithme et le code de la méthode `gainOuPerte()` soit juste.