IUT d'Orsay 2011-2012
Département Informatique DUT 2A Apprentis
Java

DS du mardi 1 février 2012

Prénom : Nom :

Durée : 2h. Indiquer clairement vos prénom et nom sur vos copies. Polycopiés et notes de cours / TD / TP autorisés. Livres non autorisés. Les programmes doivent être commentés. Les noms de variables, constantes, classes doivent être clairement choisis.

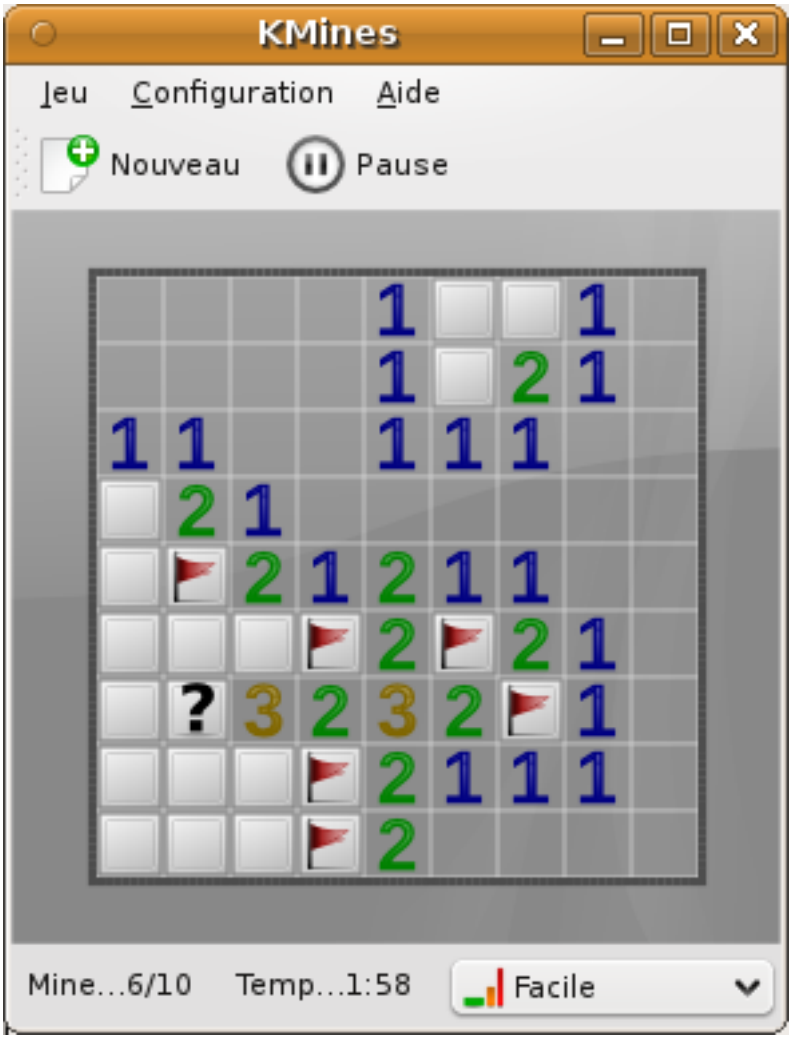
Exercice 1 (10 pt)**.** *Pour un logiciel de simulation de transport, on souhaite pourvoir manipuler des objets transportables. Un objet transportable a, au moins, un nom et une valeur. Il a de plus une méthode poids(), sans paramètre, qui doit permettre de calculer le poids de l'objet. Ce calcul est fait de manière spécifique à chaque type d'objet transportable. On ne souhaite pas pouvoir créer d'objet transportable à ce niveau de généralité.*

- *Est-il pertinent de définir* `Transportable` *comme une classe concrète ? Pourquoi ?*
- *Est-il pertinent de définir* `Transportable` *comme une classe abstraite ? Pourquoi ?*
- *Est-il pertinent de définir* `Transportable` *comme une interface ? Pourquoi ?*
- *Définir* `Transportable` *selon vos réponses aux questions précédentes (c'est-à-dire, soit comme une classe concrète, soit comme une classe abstraite, soit comme une interface*
- *Dans un programme de simulation, on souhaite manipuler des objets transportables par avion, bateau ou train. Certains objets pourront être transportés selon plusieurs moyens de transport. Par exemple, les objets d'art pourront être transportés par train et par avion. Le moyen de transport aura aussi un impact sur le coût du transport kilométrique qui dépend à la fois du poids de l'objet et du moyen de transport :*
  - *avion : 1 euros par kg ;*
  - *bateau : 0,01 euros par kg ;*
  - *train : 0,1 euros par kg.*
- *Est-il possible de créer trois classes* `TransportableParAvion`, `TransportableParBateau` *et* `TransportableParTrain` *et de définir ensuite une classe* `ObjetArt` *qui hérite en même temps des classes* `TransportableAvion` *et* `TransportableTrain` *? Pourquoi ?*
- *Implantez, avec un niveau de généralité naturel, une classe* `ObjetArt` *pour qu'elle passe le test* `JUnit` *donné page suivante. Le poids d'un objet d'art sera sa valeur divisée par 10. Si nécessaire, vous pouvez modifier* `Transportable` *et/ou créer d'autres classes.*

```
import junit.framework.TestCase;

public class ObjetArtTest extends TestCase {

    final static double PRECISION = 0.00001;
    private Transportable statue;

    protected void setUp() throws Exception {
        super.setUp();
        statue = new ObjetArt("statue ancienne", 2570.80);
    }

    public void testNom() {
        assertEquals("statue ancienne", statue.getNom());
    }

    public void testPoids() {
        assertEquals(257.08, statue.poids(), PRECISION);
    }

    public void testCoutTransportKilometriqueMin() {
        assertEquals(25.7080, statue.coutTransportKilometriqueMin(),
    }
}
```

FIGURE 1. Test JUnit de la classe ObjetArt

Exercice 2. *On souhaite implanter un jeu de démineur :*



Le but du jeu est de trouver les mines cachées sous les boutons de la grille du jeu. Un clic gauche de souris sur une case permet de révéler cette case. Si une mine se cachait sous cette case, la partie est perdue sinon, soit la case révèle un chiffre correspondant au nombre de mines adjacentes à cette case, soit la case est vide et dans ce cas les huit cases adjacentes sont découvertes et ainsi de suite pour toute case vide rencontrée. Un clic droit de souris sur une case permet d'y poser un drapeau signifiant que l'on pense qu'une mine se cache sous la case. Un second clic droit affiche un point d'interrogation signifiant que le choix est indécis, un troisième clic droit remet la case dans un état non-marqué. La partie est considérée comme gagnée lorsque que toutes les cases cachant une mine ont été marquées et toutes les cases vides découvertes.

(1) *Donner l'arbre des widgets correspondant à la vue de l'application.*

(2) *Proposer un découpage du projet en classes respectant le patron Modèle-Vue-Contrôleur. On précisera uniquement le noms des classes et interfaces en jeu (celles du projet, et celles de Java directement utilisées par le projet), leurs rôles, et leurs relations d'héritage / d'implantation.*

(3) *Donner le code complet du composant affichant la grille du démineur,* ***et seulement celle-ci****. On pourra se contenter de représenter le contenu d'une case par une chaîne de caractère adéquate ("1"-"8" pour un nombre de bombes, "D" pour un drapeau, etc).*

(4) *On souhaite pouvoir utiliser de manière interchangeable dans l'application plusieurs implantations du modèle, afin de les comparer. Vaut-il mieux utiliser pour cela une classe abstraite ou une interface ? Pourquoi ? Selon votre choix, donner le source complet de la classe abstraite ou de l'interface, décrivant toutes les méthodes publiques qui seront utilisées par le reste de l'application.*

(5) *Donner le code complet du ou des contrôleurs réagissants aux clics gauche et droit sur les cases du démineur.*