Projet de programmation

2012-2013

L2 info / math-info — 51IF2IK3

Ce document décrit plusieurs aspects de la matière 51IF2IK3 *Projet de programmation*.

Modalités de contrôle de connaissance

Cette UE est notée sur la base du contrôle continu. Vous rencontrerez toutes les semaines l'enseignant responsable de votre groupe qui évaluera le travail hebdomadaire accompli. Des documents écrits à remettre à votre enseignant et une soutenance pendant la session d'examens viendront compléter cette évaluation.

Attention ! Pas de seconde session pour cette UE.

Biblio- / Webo-graphie

Voici les url des sites et les références des livres cités dans le texte :

[1] API Java. `http://docs.oracle.com/javase/6/docs/api/`.

[2] DIDEL. `http://didel.script.jussieu.fr`.

[3] Yakov Fain. *Programmation java pour les enfants, les parents et les grands-parents*. 2004. `http://java.developpez.com/livres/javaEnfants/`.

[4] Svn. `http://usvn.script.univ-paris-diderot.fr`.

Table des matières

Chapitre I

Généralités

1 Description du cours

1.1 Modalités

Le semestre servira à réaliser un projet de programmation par groupe de 4 étudiants. Le projet n'est pas décrit dans ce document car il varie en fonction de l'enseignant et du groupe. Vous pouvez trouver les énoncés détaillés sur le site didel [2] dédié à cet enseignement.

L'évaluation de l'enseignement se fait sous forme de contrôle continu suivi d'une soutenance. Il n'y a pas de seconde session. Vous aurez par ailleurs à rendre un certain nombre de documents écrits pendant le semestre. Ces documents servent à la fois à l'évaluation et à l'avancée du projet. En particulier vous pourrez les prendre comme référence en cas de désaccord entre membres d'un groupe.

Si vous travaillez et que l'horaire du suivi est incompatible avec vos horaires de travail, joignez votre enseignant responsable le plus rapidement possible. Les coordonnées des enseignants se trouvent sur le site didel [2] dédié à la matière.

1.2 Objectif

L'objectif de ce cours est triple.

Apprendre à gérer un projet à plusieurs sur un semestre. À cette fin vous devrez remettre un certain nombre de documents écrits tout au long du semestre. Ces documents, décrits en détails dans le chapitre II, vous permettront de ne pas vous égarer dans la programmation pendant le semestre. Il s'agit des documents suivants :

- *spécification fonctionnelle* : c'est la description du produit attendu, du point de vue de l'utilisateur ; il s'agira ici de décrire à quoi ressemblera l'application que vous rendrez à la fin du semestre ;
- *spécification interne* : c'est la description du découpage de votre programme en grandes fonctionnalités et de la répartition de ces fonctionnalités dans des classes ;
- *mode d'emploi* : si votre logiciel correspond parfaitement à ce qui était prévu, il contient la spécification fonctionnelle, sinon il contient sa mise à jour en fonction de la réalité ; ce document devra par ailleurs décrire les diverses étapes à suivre par un utilisateur pour obtenir un logiciel fonctionnel.
- javadoc : documentation de votre code.

Vous devrez également avancer régulièrement dans votre travail de programmation.

Apprendre à trouver ce dont on a besoin. Ce cours a une forme non classique par rapport aux autres enseignements que vous avez suivis et que vous aurez l'occasion de suivre pendant votre licence d'informatique ou de mathématiques-informatique à l'Université Paris Diderot. En effet, vous n'aurez pas des séances de cours et des séances d'exercices tout au long du semestre ; les connaissances acquises à la fin de cet enseignement sont celles que vous irez chercher par vous-mêmes. L'enseignant est là pour vous guider dans vos choix et apporter les réponses que vous n'aurez pas trouvées par ailleurs, voire vous indiquer où trouver ces réponses. Chaque groupe aura donc acquis *a priori* des connaissances de programmation différentes à la fin du semestre. L'apport essentiel concernant la programmation de ce cours est donc la capacité que vous aurez acquise pour chercher et trouver des réponses.

Acquérir plus d'aisance pour programmer. Pour la première fois (pour la plupart d'entre vous), vous allez écrire un "gros" programme et non juste une fonction ou une classe qui répond à une question précise. En plus de la partie conception déjà soulignée, cela implique que vous soyez capables d'écrire du code simple sans regarder le cours pour chaque boucle et d'utiliser des classes existantes de l'API java en vous basant sur la lecture de la javadoc de ces classes.

1.3 Outils informatiques mis à votre disposition

Outre le site didel [2] dédié à l'enseignement, vous avez accès à un serveur de version svn [4] qui vous permettra de travailler séparément ou ensemble et de pouvoir à tout moment déposer ou récupérer une version à jour de votre travail. Ce site permettra également à l'enseignant qui vous suit de voir la progression de votre travail. Une brève description du fonctionnement d'un serveur svn se trouve au chapitre III.

Les divers documents à rendre seront rendus *via* votre dépôt svn ou sur didel.

Par ailleurs, les forums didel dédiés à chaque projet peuvent vous permettre d'échanger des questions/réponses d'ordre général avec les autres groupes faisant le même projet.

2 Pendant le semestre

2.1 Séances de suivi

Le créneau horaire *cours* de votre emploi du temps ne sera un cours que pour la première séance durant laquelle votre enseignant vous décrira le(s) projet(s) et vous fera un rapide cours concernant des notions particulières liées au(x) projet(s) proposé(s).

Ensuite et pendant toute la durée du semestre, ce créneau horaire servira aux séances de suivi : l'enseignant recevra chaque groupe de façon hebdomadaire pour faire le point sur les documents rendus et/ou l'avancement du projet. Pour cela il est évidemment indispensable de respecter les dates limites de rendu fixées par votre enseignant.

Pour un groupe donné la séance de suivi dure 15 minutes à une demi-heure, en fonction des effectifs. Vous pouvez profiter du reste de ce créneau horaire pour travailler ensemble : le script fournit des machines en libre-service et pour ceux qui n'ont pas d'ordinateur, l'emprunt de netbook est possible également auprès du script.

Chaque semaine un membre du groupe sera *chef de projet* : il devra s'assurer que le travail est réparti correctement entre les membres du groupe, que chacun fait ce qu'il doit faire et il arbitrera en cas de conflit. Par ailleurs il est chargé de fournir à l'enseignant responsable une fiche de suivi hebdomadaire remplie *avant* la séance de suivi (fiche disponible en annexe et sur didel).

2.2 Libre-service

Votre emploi du temps prévoit deux créneaux horaires de 2 heures hebdomadaires de *travaux pratiques*. Il

– 5 minutes de présentation globale comprenant en particulier la présentation du résultat final,
– 5 minutes par étudiant pour expliquer un point plus précis du projet.
Les salles de TP du script ont un vidéoprojecteur relié à un des ordinateurs de la salle ou que l'on peut relier à un portable (nous apportons un cable VGA *sur demande* mais veillez à vous assurer que vous pouvez ensuite connecter votre machine à ce cable).

La soutenance commencera par la création d'un dépôt local pour votre projet et sa mise à jour. Toutes les démonstrations se feront sur la base de ce dépôt. Le jour de la soutenance vous pouvez ajouter des fichiers de présentation ou de démonstration.

A la fin de cette soutenance, les enseignants qui y assistent doivent avoir une idée des fonctionnalités de votre programme, de sa structure générale et des algorithmes utilisés. Les enseignants doivent être en mesure d'utiliser votre programme. Il vous est demandé par ailleurs d'être très précis sur le découpage du travail (qui a fait quoi).

3.3 La note

La note tient compte (l'ordre des points suivants n'est pas significatif) :
– du travail pendant le semestre,
– des divers documents rendus,
– de la gestion du travail collectif pendant le semestre,
– du résultat final,
– de la qualité du code,
– de la soutenance
– et de points particuliers relatifs à votre sujet.

Chapitre II

Documents à rendre

Les dates précises de rendu des documents seront notées sur didel [2]. Il faut les respecter pour que l'enseignant puisse vous faire un retour lors de la session de suivi qui suit. Il est possible que votre enseignant vous demande de réécrire tout ou partie du document.

Le calendrier général est le suivant :
– spécification fonctionnelle : pour la séance 1,
– spécification interne : pour la séance 2,
– mode d'emploi : 2 semaines avant la fin du semestre,
– javadoc : au fur et à mesure du code.

1 Spécification fonctionnelle / cahier des charges

La *spécification fonctionnelle* (ou *cahier des charges*) est le document qui décrit de manière précise le produit fini vu de l'extérieur. Il ne contient en principe aucun détail technique sauf éventuellement des contraintes de fonctionnement par exemple la spécification du système sous lequel doit tourner le logiciel ou un format avec lequel il doit être compatible.

Dans une vie théorique, le cahier des charges est remis par le client (c'est-à-dire celui qui commande le produit) et décrit parfaitement ce que le client souhaite. Dans la vie réelle, le client a en général du mal à spécifier complètement ce qu'il désire, ne serait-ce que parce qu'il ne pense pas à tous les détails ; il est donc souvent aidé par une personne qui a plus l'habitude de ce genre de documents et le cahier des charges peut donner lieu à des allers-retours entre client et équipe de programmation. Dans votre vie d'IK3, c'est vous qui allez écrire ce document. Vous essaierez ensuite de vous y conformer pendant le semestre. Plus le document est précis et exhaustif, plus il vous sera facile de travailler séparément. La présence d'ambiguïtés ou de trous dans ce document peut retarder votre travail de programmation et occasionner des va-et-vient de code si tous les membres du groupe n'ont pas la même interprétation d'une fonctionnalité.

Que doit-on mettre dans un tel document ? Supposons qu'on veuille écrire une application qui affiche une horloge (à l'heure) sur le fond d'écran. Voici certaines des questions auxquelles ce document devra répondre :
– à quoi doit ressembler l'horloge ? affichage numérique, par aiguille ? (on peut parfaitement imaginer un dessin pour la réponse à cette question)
– l'utilisateur peut-il modifier l'affichage ? comment ?
– l'heure est-elle réglée sur l'heure système ? sinon sur quoi ? comment la modifier ?
– peut-on régler l'heure système à travers cette application ?
– l'application comporte-t-elle un réveil ? comment le régler ? comment régler la "sonnerie" ?
– a-t-on la possibilité de mettre l'horloge au premier plan, devant les autres fenêtres ouvertes ? (on peut juger que cette question est hors sujet car on veut afficher l'horloge sur le fond d'écran)

– peut-on modifier la taille de l'horloge ?

Evidemment, plus l'application à réaliser est complexe, plus le cahier des charges sera long.

Ce document vous servira tout au long du semestre à développer votre application. Vous pouvez être amenés à le modifier si la spécification ne semble pas réaliste par rapport à vos capacités de programmeur ou à le compléter si vous aviez oublié certaines considérations. Il est important qu'il soit à jour car vous vous y réfèrerez pour travailler séparément.

2 Spécification interne

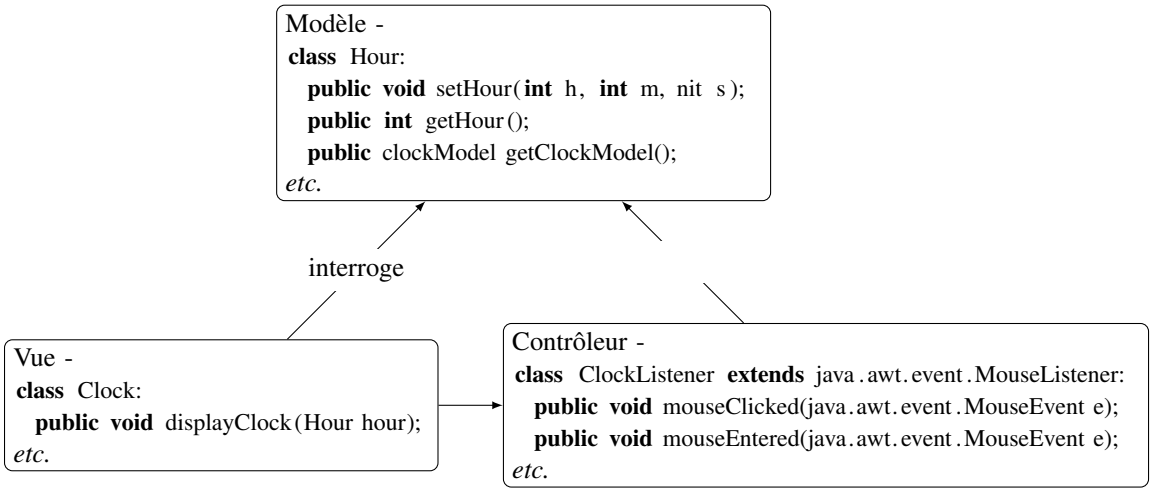
La *spécification interne* décrit les grandes fonctionnalités de votre programme et son architecture.

Reprenons l'exemple de l'horloge de tout à l'heure. A cet exemple peuvent être associées trois familles de fonctionnalités :

le modèle qui regroupe toutes les données contractuelles (heure courante, modèle d'horloge à afficher, etc.) ainsi que les méthodes pour les traiter (modifier l'heure courante, modifier l'apparence de l'horloge, etc.) ;

la vue qui effectue l'affichage en cohérence avec le modèle et transmet les actions de l'utilisateur au contrôleur ; c'est l'interface graphique avec laquelle l'utilisateur interagit ;

le contrôleur qui gère les événements pour mettre à jour le modèle (en fonction des solicitations de l'utilisateur) et le synchroniser avec la vue.



4 javadoc

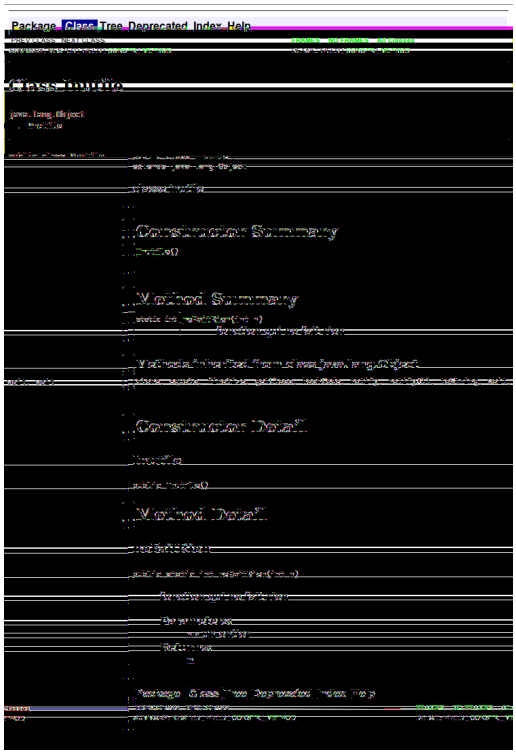
La javadoc est un format de documentation de code java qui permet de générer automatiquement des fichiers de documentation au même format html que la documentation officielle java [1].

Cette documentation permet d'accélérer la programmation car on n'a plus besoin de fouiller dans le code pour savoir à quoi sert une classe ou une méthode.

Écrire la documentation au fur et à mesure est plus rapide et nettement plus utile que de l'écrire à la fin. En effet, au moment où l'on crée une classe ou où l'on écrit une méthode on sait exactement à quoi ça sert et quand on veut la réutiliser ou utiliser du code écrit par un autre membre du groupe, on est immédiatement fonctionnel.

Ci-dessous un exemple de documentation au format javadoc et ci-contre le résultat graphique produit (en lançant la commande javadoc sur le source java) :

```
/
   classe inutile
  /
public class Inutile{
    /
       fonction qui ne fait rien
       @param n un entier quelconque
       @return n
      /
    public static int neFaitRien(int n){
        return n;
    }
}
```



Chapitre III

subversion – svn

1 Contexte d'utilisation de l'outil

Quand on développe à plusieurs, il est important que chacun puisse avoir accès à tout moment à tous les fichiers et puisse les modifier. Si plusieurs personnes travaillent à partir de la même version d'un fichier, il faut pouvoir *synchroniser* les modifications faites par chacun pour produire une version unique.

Un *outil de gestion de versions* sert à *automatiser*, tant que possible, ce travail de synchronisation. Il en existe plusieurs, pour IK3 nous mettons à votre disposition un serveur subversion [4] (svn de son petit nom).

2 Principe de fonctionnement basique

subversion est un gestionnaire de versions dit centralisé. Cela signifie qu'il existe un serveur qui stocke l'ensemble des versions du code source : c'est le *dépôt*. C'est en interagissant avec ce serveur que le programmeur peut :
– soit se mettre à jour en important les modifications publiées par ses partenaires depuis le serveur vers sa *copie locale* ;
– soit publier ses propres modifications depuis sa copie locale vers le serveur.

Lorsque deux utilisateurs travaillent en même temps sur un même fichier, deux cas de figure peuvent surgir :

1. ils ne modifient pas les mêmes parties d'un fichier : dans ce cas le gestionnaire de versions intègre les modifications de chacun ;
2. ils modifient des parties en commun : dans ce cas apparaît un *conflit* signalé par le gestionnaire de versions ; les utilisateurs doivent *résoudre* ce conflit (en effet, le gestionnaire de versions ne peut pas décider tout seul quelles sont les bonnes modifications).

Un gestionnaire de versions propose d'autres fonctionnalités comme par exemple la récupération d'une version antérieure d'un fichier. Cependant, l'utilisation basique décrite ici devrait vous suffir pour collaborer sur votre projet.

3 Utilisations principales

subversion se présente sous la forme d'un programme utilisable en ligne de commande. Dans cette section, on décrit les utilisations basiques auxquelles vous serez confrontées.

Initialisation d'une copie locale À partir des informations d'authentification et de l'URL qui vous auront été données, vous pouvez initialiser une copie locale du projet sur votre compte.

TABLE 1 – à lancer la première fois

`svn checkout -username login http://url`	initialisation d'une copie locale

Pensez à remplacer `login` par votre login ENT et `url` par l'url qui vous aura été donné par l'enseignant qui vous suit.

Gestion des fichiers à publier Un répertoire de votre copie locale peut contenir à la fois des fichiers que vous souhaitez publier et d'autres non (cf. section 4), c'est pourquoi subversion ne prend en compte que les fichiers explicitement intégrés au système par vos soins.

La table 2 récapitule diverses fonctions de base concernant les fichiers.

TABLE 2 – gestion des fichiers (maj : mise à jour)

`svn add fichier`	déclare l'ajout d'un fichier pour la prochaine maj
`svn rm fichier`	déclare la suppression du fichier pour la prochaine maj
`svn mv ancien-nom nouveau-nom`	modifie le nom du fichier pour la prochaine maj

Ces commandes fonctionnent aussi bien sur les fichiers que les répertoires. Dans ce dernier cas, c'est le contenu du répertoire qui est ciblé par l'action de la commande.

Mise à jour de la copie locale Vos fichiers sont mis à jour automatiquement en fonction des modifications publiés par vos partenaires. Si un conflit est détecté, subversion produit un message d'erreur en indiquant les fichiers en conflit.

TABLE 3 – au début de chaque séance de travail

`svn update`	récupérer la version du dépôt sur la copie locale

Publication des modifications locales Vos modifications sont transférées dans le dépôt.

TABLE 4 – à la fin de chaque séance de travail

`svn commit -m "commentaire"`	publier les modifications locales

Le commentaire vous permet de décrire succintement vos modifications pour la documentation de l'historique du projet.

L'opération de publication peut échouer si une nouvelle version a été publiée par un de vos partenaires depuis votre dernière mise à jour. Dans ce cas, vous devez d'abord vous mettre à jour à l'aide de la commande `svn update` décrite dans le paragraphe précédent.

4 Quels fichiers mettre dans son dépôt svn ?

Le gestionnaire de versions subversion ne conserve pas les versions successives, mais les modifications faite d'une version à la suivante, pour limiter la place mémoire utilisée. Pour cette raison, il faut éviter de

versionner des fichiers binaires car subversion ne saura pas extraire les différences [1] et sauvegardera les versions successives complètes.

Ainsi, on versionne dans le serveur :
- des sources java,
- des fichiers texte,
- des sources LaTeX.

On ne versionne pas dans le dépôt :
- du *bytecode*,
- un fichier ods,
- un fichier doc.

1. Tout comme la commande diff n'est pas très utile sur des fichiers binaires.

Chapitre IV

Introduction à la notion d'objets, de classes et d'interfaces

Le but de ce chapitre n'est pas de vous fournir un cours de programmation objet, mais uniquement des éléments de base vous permettant de modéliser vos projets de manière un peu réfléchie.

Pour plus de détails, [3] est un livre d'introduction particulièrement simple.

Dans ce qui suit, tout le code n'est pas écrit systématiquement, afin d'alléger le texte. Des commentaires de la forme `/*...*/` signifie qu'il manque du code. En principe les noms des méthodes sont suffisament explicites pour comprendre ce qu'elles font.

1 Objets et classes

Une *classe* est un modèle de donnée qui permet de regrouper des valeurs (les *attributs* de la classe) et des actions applicables à ces valeurs (les *méthodes* de la classe).

Un *objet* est une *instance* de cette classe s'il en possède les attributs et les méthodes.

Exemple 1. On représente un rectangle comme une instance de la classe Rectangle suivante

```
import java.awt.Point;

/**
 * representation d'un rectangle donne par son coin superieur gauche
 * et ses dimensions
 */
public class Rectangle {
    private int x, y;
    private int width, height;

    public Rectangle(){
        this(0,0,0,0);
    }

    public Rectangle(int x, int y, int width, int height){
        super();
        this.x = x; this.y = y;
        this.width = width; this.height = height;
    }
```

```
    public Rectangle(Rectangle r){
22      this(r.x, r.y, r.width, r.height);
    }

    public boolean contains(Point p){ /*...*/ }

    public Rectangle createIntersection(Rectangle r){ /*...*/ }
28  }
```

La classe ainsi définie permet de tester si un point est à l'intérieur d'un rectangle et de calculer l'intersection de deux rectangles.

On remarque dans les constructeurs l'appel à **super** (...) : c'est l'appel au constructeur de la super-classe (cf. section 2). Pour appeler un autre constructeur de la même classe, on utilise **this** (...) .

2 Héritage

On veut maintenant représenter des rectangles colorés. On peut parfaitement ajouter un attribut à notre classe Rectangle précédemment définie, mais cette méthode présente des inconvénients :
- soit on modifie le code (qui fonctionne) de la classe Rectangle :
  - on risque d'introduire des erreurs,
  - la classe ainsi réécrite doit rester compatible avec d'autres classes qui auraient pu se servir de celle-ci,
  - on remarque que la classe standard java.awt.Rectangle existe et qu'elle fait ce que l'on souhaite (et plein d'autres choses).
- soit on copie le contenu de cette classe dans une autre à laquelle on a ajouté la couleur :
  - il faut alors maintenir deux classes,
  - on ne peut pas considérer un rectangle coloré comme un rectangle.

Il est clairement plus malin d'utiliser la classe java.awt.Rectangle. Cela rend complètement caduque l'idée de compléter le code de la classe elle-même : on ne modifie pas les librairies standards du langage.

Pour créer une classe représentant des rectangles colorés, on utilise le mécanisme d'*héritage* offert par java, en *étendant* la classe Rectangle :

```
    import java.awt.Rectangle;
2   import java.awt.Color;

4   public class ColoredRectangle extends Rectangle {
        private Color color;

        public ColoredRectangle(){
8           super(); this.color = Color.BLACK;
        }

        public ColoredRectangle(int x, int y, int width, int height, Color color){
12          super(x, y, width, height);
            this.color = color;
14      }

16      public ColoredRectangle(Rectangle r, Color color){
            super(r); this.color = color;
18      }

20      private ColoredRectangle(Rectangle r){
            this(r, Color.BLACK);
```

```
22      }
}
```

Les instances de cette classe possèdent les attributs x, y, width et height hérités de java.awt.Rectangle (et qui sont **public** dans cette dernière) et on peut leur appliquer toutes les méthodes (**public** ou **protected**) de leur classe mère.

On utilise de façon équivalente les expressions suivantes :
– ColoredRectangle *étend / hérite de / est une sous-classe de / est une classe fille de* Rectangle
– Rectangle est *étendue par / la super-classe de / la classe mère de* ColoredRectangle
On peut invoquer une méthode de la classe Rectangle à travers une instance de la classe ColoredRectangle car un tel objet est également une instance de la classe Rectangle.

Bien entendu il est parfois opportun de modifier ou compléter certaines méthodes de la classe Rectangle :
– soit en les *surchargeant*, c'est-à-dire en réutilisant le nom de la méthode, mais en modifiant sa signature, par exemple on pourrait penser que l'intersection entre deux instances de ColoredRectangle est une instance de ColoredRectangle dont la couleur est intermédiaire entre les couleurs des deux rectangles de départ :

Dans la classe ColoredRectangle

```
public ColoredRectangle createIntersection(ColoredRectangle r){
2       Rectangle r2 = (Rectangle) r;
        ColoredRectangle res = new ColoredRectangle(createIntersection(r2));
4       res.color = this.averageColor(r.color);
}

private Color averageColor(Color c){ /*...*/ }
```

On remarque que la méthode Rectangle createIntersection (Rectangle r) est encore utilisable directement (elle est d'ailleurs utilisée dans la définition de la surcharge), car elle n'a pas la même signature que la surcharge que l'on vient de définir.
En pratique la méthode utilisée est celle correspondant à la classe la plus fine parmi les méthodes utilisables.
– soit en les *redéfinissant*, c'est-à-dire en donnant une nouvelle définition de la méthode (avec la même signature et le même type de retour), par exemple la méthode equals peut être redéfinie de la manière suivante :

Dans la classe ColoredRectangle

```
public boolean equals(Object o){
2       if (o==this) return true;
        if (!(o instanceof ColoredRectangle)) return false;
4       return super.equals(o) && this.color==((ColoredRectangle)o).color;
}
```

Ici on a besoin de la méthode définie dans la classe mère, on utilise le préfixe **super**. pour l'invoquer.
Notez que la méthode equals est héritée de la classe Object : toutes les classes la possèdent.

3 Interfaces

Une *interface* sert à spécifier un comportement.

Prenons l'interface suivante :

```
import java.awt.Graphics2D;

interface GeometricShape {
```

```
4       void drawMe(Graphics2D g);
}
```

On n'y voit que des déclarations de méthodes : on dit que ces méthodes sont *abstraites* car elles ne sont pas définies. Ces en-têtes sont données sans spécification d'accessibilité car les méthodes d'une interface sont toujours publiques.

On ne peut pas instancier directement une interface (c'est-à-dire faire un **new** dessus) puisque la machine virtuelle ne saurait pas comment exécuter les méthodes.

Toute classe *implémentant* une interface doit suivre le comportement de cette interface, c'est-à-dire définir ses méthodes. On pourrait par exemple compléter la classe ColoredRectangle par :

```
public class ColoredRectangle implements GeometricShape {
2       /*...*/
        public void drawMe(Graphics2D g){
4           Color c = g.getColor();
            g.setColor(this.color);
6           g.fill(this);
            g.setColor(c);
8       }
}
```

Remarquez la ligne 6 : on invoque la méthode fill à travers une instance de la classe Graphics2D. En effet, le type de l'argument de cette méthode est l'interface java.awt.Shape qui est implémentée par la classe Rectangle qui est elle-même étendue par ColoredRectangle, donc la classe ColoredRectangle implémente l'interface Shape par héritage :

$$\textit{Shape} \dashleftarrow \text{Rectangle} \longleftarrow \text{ColoredRectangle}$$

Supposons qu'on crée une classe Triangle qui implémente également GeometricShape. On pourrait alors écrire le programme suivant :

```
class Draw {
2       public static void main(String[] args){
            Scanner sc = new Scanner(System.in);
4           int response;
            GeometricShape shape;
6           Graphics2D g = /*...*/;

8           System.out.println("quelle forme dois-je dessiner ?");
            System.out.println("rectangle rouge (0), triangle (1)");
10          response = sc.nextInt();
            switch(response){
12          case 0: shape = new ColoredRectangle(10,10,50,80,Color.RED);
              break;
14          case 1: shape = new Triangle(/*...*/);
              break;
16          default: shape = new Rectangle(10,10,50,80);
            }

            shape.drawMe(g);
20      }
}
```

Une bonne pratique en java est de privilégier les déclarations de variables et les types de paramètres avec des interfaces, ce qui permet une plus grande souplesse dans la réutilisation du code.

4 Classes abstraites

On peut également avoir des classes dont certaines méthodes sont abstraites (mot-clef **abstract**). Ces classes sont elles-mêmes définies comme abstraites et ne peuvent être instanciées directement.

Revenons à notre exemple de rectangle coloré et supposons qu'il puisse subir des déformations :

```
public abstract class DistortedColoredRectangle extends ColoredRectangle {
    abstract ColoredRectangle distortRectangle();

    public void drawMe(Graphics2D g){
        this.distortRectangle().drawMe(g);
    }
}
```

La classe DistortedColoredRectangle étend la classe ColoredRectangle. Elle redéfinit la méthode **void** drawMe(Graphics2D g) et celle-ci fait appel à une méthode abstraite ColoredRectangle distortRectangle(). Les sous-classes non abstraites de DistortedColoredRectangle devront définir cette méthode.

Par exemple :

```
public class NightVisionColoredRectangle extends DistortedColoredRectangle {
    public ColoredRectangle distortRectangle(){
        /*...*/ //renvoie un rectangle plus fonce
    }
}
```

ou encore :

```
public class MagnifyingColoredRectangle extends DistortedColoredRectangle {
    public ColoredRectangle distortRectangle(){
        /*...*/ //renvoie un rectangle plus grand, de meme centre
    }
}
```

Comme précédemment, on pourra déclarer une variable ou un paramètre de type DistortedColoredRectangle et pour lui donner une valeur instancier un objet de type NightVisionColoredRectangle ou MagnifyingColoredRectangle selon les circonstances. Le choix entre ces deux instanciations reste transparent pour le reste du code.

Fiche d'évaluation hebdomadaire

projet :

date :

membres du groupe (souligner le chef de projet de la semaine) :

1.
2.
3.
4.

Accompli cette semaine

Difficultés rencontrées

A faire pour la semaine prochaine