

Nom :

Prénom :

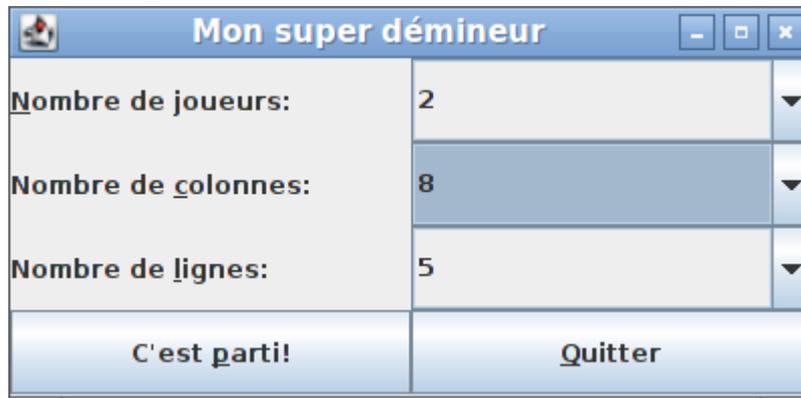
## DS POO 2017-2018

### 1ère session

L'objectif du sujet est de réaliser un programme multijoueur ressemblant au jeu du démineur.

#### 1 Fonctionnement du jeu :

1) une partie commence en choisissant un nombre de lignes et de colonnes pour la grille de jeu, ainsi que le nombre de joueurs (figure 1).

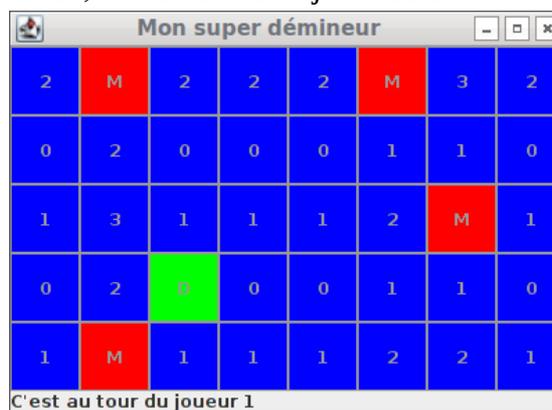


Mon super démineur	
Nombre de joueurs:	2
Nombre de colonnes:	8
Nombre de lignes:	5
C'est parti!	
Quitter	

2) si l'utilisateur clique sur « C'est parti », le programme crée l'aire de jeu en fonction des paramètres renseignés ; chaque case est un JButton qui n'affiche aucun texte.

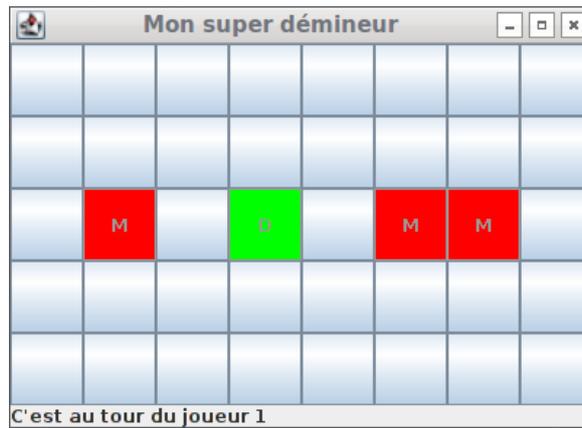


Voici, pour information, la même aire de jeu avec cette fois-ci les cases affichées :



3) les joueurs jouent à tour de rôle en cliquant sur une case qui n'a pas été encore jouée. La case est alors révélée et dessinée, et en fonction du type de case trois comportements sont possibles :

- a) les cases de type « Demine », représentées en vert, révèlent toutes les cases de type mine de la ligne à laquelle la case cliquée appartient :



- b) les cases de type « Info », en bleu, indiquent le nombre de mines présentes sur la colonne et la ligne de la case info



- c) les cases de type « Mine », en rouge, révèlent l'aire de jeu et font perdre la partie au joueur qui les a activé :



Un joueur gagne si il révèle la dernière case non visible de l'aire de jeu. Un joueur perd si il révèle une mine.

Le code partiel de ce programme est donné en fin de sujet, il vous appartient de le compléter en fonction des questions. Les questions sont indépendantes mais il est préférable de les traiter dans l'ordre.

## 2 Compréhension du code (8 points)

Vous disposez en fin de sujet du code du programme qu'il conviendra de compléter.

## 2.1 Comment sont organisées physiquement les classes, et pourquoi ? (0,5 pt)

Les classes sont organisées dans deux packages : le premier, `model`, contient les classes métiers de l'application.

Le deuxième, `ihm`, contient les classes de l'interface graphique.

Elles sont organisées de cette manière afin de séparer la partie modèle de la partie interface graphique.

## 2.2 Que sont les attributs `public static final` des classes `Plateau` et `Jeu` ? Quelle est leur utilité ? Est-ce une bonne idée de les déclarer `public` ? (1 pt)

Ce sont des constantes.

Elles permettent de définir les paramètres fixes du programme.

Normalement, un attribut doit être `private` ou `protected`, mais dans le cas particulier de constantes (non modifiables), il est au contraire plus pratique de les déclarer `public` (comme par exemple `Math.Pi` de la classe `Math`, ou `BorderLayout.NORTH`, etc.).

## 2.3 La `Map` ligne 137 du code sert au tirage aléatoire des types de case ? Pourquoi la méthode `createMap()` est-elle `static` ? L'initialisation de la `Map` est-elle possible de cette manière ? Si non, proposez une autre manière d'initialiser cette `Map` (pas besoin de coder). (1 pt)

`createMap` est statique car elle sert à initialiser une variable statique.

De plus, c'est une fonction utilitaire qui ne dépend pas des attributs non-statiques de la classe où elle est définie.

La `Map` est statique et c'est une constante : elle doit être initialisée lors de sa déclaration, par une méthode statique. L'initialisation est donc possible de cette manière.

## 2.4 Regardez les méthodes `init()` et `tirage()` de la classe `Plateau` (lignes 103 à 126). Expliquer le fonctionnement de ces deux méthodes et à quoi elle servent (2 pt).

La méthode `init` parcourt une liste de liste : soit une structure bi-dimensionnelle représentant le plateau de jeu.

Pour chaque élément du plateau, elle fait appel à la méthode `tirage` qui lui renvoie une instance de `Case` obtenue par un processus de tirage aléatoire.

Cette instance est ensuite clonée, et on lui affecte ses coordonnées.

La méthode `tirage` obtient un nombre aléatoire compris entre 0 et 1, le multiplie par 100 et le cast ensuite en entier.

On obtient donc un nombre aléatoire compris entre 0 et 99.

`Tirage` parcourt ensuite la `map` contenant une instance de chaque type de `Case` (`Mine`, `Demine`, ...) en clef, et compare la valeur associée à la clef au nombre tiré : si cette valeur est inférieure à la clef, alors la clef est retournée : il est donc indispensable que la `map` contienne une clef dont la valeur est associée à 100 afin d'être certain qu'une clef sera retournée.

En résumé, ces deux méthodes permettent l'initialisation du plateau de jeu de manière aléatoire.

## 2.5 Pourquoi fait-on appel à setCase ligne 121 ? Est-ce indispensable?(1 pt)

setCase est un mutateur permettant d'attribuer ses coordonnées à une case.

Une bonne pratique de la programmation objet consiste à utiliser des mutateurs.

Dans le cas particulier de la ligne 121, la case a été clonée à partir d'une instance d'un héritier de case construite avec un constructeur par défaut (dans la map) : les coordonnées de la case sont donc celles par défaut.

Il est donc indispensable d'affecter les coordonnées réelles de la case, sinon toutes les cases auraient les même coordonnées par défaut.

## 2.6 Lors de la création d'un Plateau avec new Plateau(5,5), combien d'instances de Case sont créées ? (0,5 pt)

$5*5=25$  instances (liste de 5 listes de 5 cases)

## 2.7 (1 pt) Dans tirage(), quelle est la probabilité en pourcentage de :

- renvoyer une instance de Demine ?

5

- renvoyer une instance de Mine ?

15 ( $p=20-5$ ), il faut penser à retirer la probabilité de tirer une instance de Demine

## 2.8 Pourquoi utilise-t-on une LinkedHashMap et pas une HashMap ligne 156 ? (1 pt)

Le tri des clef d'une HashMap se fait à partir du hashcode de la clef, pas de son ordre d'insertion.

Comme les probabilités de tirage doivent être classées par ordre croissant, il faut une structure qui conserve l'ordre d'insertion des clefs : une LinkedHashMap

## 3 Codage

Les espaces laissés vides sont à compléter. Leur taille devrait suffire pour contenir votre code.

Parfois, seule une petite partie de cet espace est nécessaire.

Il faut respecter les import déclarés (pas d'autre import nécessaires).

## 4 Codage du Modèle (20 points)

Le modèle contient les classes qui modélisent le problème sans gérer la partie IHM.

### 4.1 compléter les lignes 6, 37, 51, 67 pour représenter la hiérarchie de Case. (1 pt)

Attention : Certains espaces peuvent rester vides.

Demine, Mine et Info héritent de Case

Case est abstract

Dans Case, les coordonnées d'une case sont représentées à l'aide de la classe Point de l'API java (voir extrait de la javadoc de Point **sur deux pages** en fin de sujet).

### 4.2 Quel est l'intérêt, dans la philosophie objet, d'utiliser cette classe ? (1 pt)

Réutiliser du code existant, confier la responsabilité de la gestion des coordonnées à une classe existante conçue pour prendre en charge cette responsabilité

#### 4.3 Est-ce la classe Point respecte les bonnes pratiques de conception objet ? Pourquoi ? (1 pt)

Non : ses attributs x et y sont publics, ce qui est contraire au respect de l'encapsulation.  
On n'est pas dans le cas de constantes cette fois-ci.

#### 4.4 Codez la méthode setCase de la classe Case (1 pt)

Les coordonnées x et y doivent être valides (  $0 \leq x < \text{longueur}$  ,  $0 \leq y < \text{largeur}$  ).  
Si ce n'est pas le cas, les coordonnées (0,0) seront utilisées.

longueur et largeur viennent de Plateau

#### 4.5 Complétez les constructeurs de Mine, Demine et Info (2 pt)

Pensez à gérer les couleurs des Cases

Il faut utiliser correctement le mot-clef super

#### 4.6 Coder les méthodes toString() des Case (2 pt)

Les méthodes toString() des Cases ont le comportement suivant :

- 1) si la case n'est pas visible, renvoie « ? »
- 2) si la case est visible, renvoie la première lettre du nom de la classe (D pour Demine, M pour Mine...)
- 3) cas particulier : pour Info, affiche la valeur de l'attribut nbMines

Rappels :

- 1) `monObjet.getClass().getSimpleName()` renvoie une String contenant le nom de la classe.
- 2) extrait de la javadoc de String :

[String](#) `substring(int beginIndex)`  
Returns a string that is a substring of this string.

[String](#) `substring(int beginIndex, int endIndex)`  
Returns a string that is a substring of this string.

le comportement général peut être codé dans Case. En revanche, nbMine n'est accessible que dans la classe Mine, il faut donc redéfinir une nouvelle fois toString dans Mine

#### 4.7 Clonage (3 pt)

Mettez en place le clonage de Case, en obligeant les héritiers de Case à redéfinir la méthode clone() d'Object. Pensez bien à toutes les modifications à apporter à l'ensemble des classes de la hiérarchie de Case.

Case implémente clonable, mais ne donne pas le code de clone (possible si Case est abstract)

Les héritiers sont donc obligés de donner un code pour clone, sinon ils devront être déclarés abstract ce qui n'est pas souhaitable !

#### 4.8 Codez les méthodes boolean processAction(ArrayList<ArrayList<Case>> cases) (5 pt).

Le paramètre **cases** représente les cases de l'aire de jeu.

Cette méthode est appelée lorsqu'une case doit être révélée (par exemple lorsqu'on a cliqué dessus) et applique les algorithmes suivants présentés en introduction (section 1.3).

Elle renvoie **true** si la partie est terminée, **false** sinon.

Attention : vous devez :

- utiliser un itérateur dans Demine ;
- mettre à jour l'attribut nbMines dans Info ( ;
- lorsqu'une Mine est activée, il faut révéler toutes les cases, donc bien penser à appeler processAction pour chacune des cases de **cases** qui ne sont pas des mines (pour par exemple avoir un affichage cohérent des cases Info).

#### 4.9 Codez les constructeurs de Jeu (2 pt)

Vous devez être le plus concis possible.

Jeu a la responsabilité de créer un plateau et d'initialiser le nombre de joueurs.

Pensez à vérifier que les arguments sont valides.

utiliser le mot-clef `this` pour ne pas avoir à dupliquer le code !

#### 4.10 Codez la méthode `nextJoueur()` dans `Jeu` (1 pt)

En java, l'opérateur `%` renvoie le reste de la division entière de deux entiers.

Par exemple :

$5 \% 6 = 5$        $5 \% 5 = 0$        $5 \% 4 = 1$

### 5 Codage de l'interface graphique (Fenetre, 16 points)

#### 5.1 Codez l'intégralité de la méthode `initDialogue` (6 points)

Consultez la documentation sur `JComboBox`, `JLabel` et `JButton`.

Vous devez obtenir le même rendu que dans la partie 1.1 de Information.

Penser aux listener (méthode « on the fly »).

#### 5.2 Codez la boucle de la méthode `init()` et de `ListenerBt` (10 points)

Cette boucle crée des `JButton` et les place dans chaque case de la grille.

Il vous faudra utiliser un Listener externe (`ListenerBt`) dont vous devez compléter le code.

Dès qu'un bouton est cliqué, il doit être désactivé.

Attention, le Listener est externe (voir cours pour cette technique) car le bouton a besoin de connaître la case pour appeler `processAction`

### 6 Évolutions (5 points)

On vous demande de prévoir un nouveau type de case : `DemineVert`, qui à le même comportement que `Demine` mais pour une colonne.

Quels sont les aménagements à apporter au code ? Pensez bien à **toutes** les classes impactées et à proposer une solution la plus respectueuse des principes de conception objet.

En fonction de ce que vous proposez :

Créer une classe `DemineVert`

Renommer `Demine` en `DemineHor`

Créer une classe intermédiaire `Demine` dont héritent `DemineVert` et `DemineHor`

Redéfinir `toString` dans `DemineHor` et `DemineVert` car comme le nom de ces classes commence par la même lettre, le `toString` de `Case` ne convient plus.

Mettre à jour la Map de tirage aléatoire.

Créer une interface `Demine`, `DemineHor` et `DemineVert` implémentent cette interface

Créer une interface « `ComportementCase` », `Case` implémente `ComportementCase` et oblige ses héritiers à définir `processAction` (c'est la solution la plus « POO »).

## 7 Code à compléter

```
1 package ds4eti2018_1S.modele;
2 import java.awt.Color;
3 import java.awt.Point;
4 import java.util.ArrayList;
5
6 public class Case
7 {
8     private Point coord;
9     private boolean visible = false;
10    private Color couleur;
11
12    public Case(){this(0,0, Color.black);}
13    public Case(int x, int y, Color couleur){
14        coord = new Point(x,y);
15        this.couleur=couleur;
16    }
17
18    protected int getX(){return coord.x;}
19    protected int getY(){return coord.y;}
20
21    public Color getCouleur() {return couleur;}
22    public boolean isVisible() {return visible;}
23    public void setVisible(boolean visible) {this.visible = visible;}
24
25    public void setCase(int x, int y){
26    }
27    public String toString(){
28
29    }
30    public boolean processAction(ArrayList<ArrayList<Case>> cases){
31        this.setVisible(true);
32        return false;
33    }
34 }
```

```
34 package ds4eti2018_1S.modele;  
35 import java.awt.Color;  
36 import java.util.ArrayList;  
37 public class Mine  
38 {  
39     public Mine(){
```

```
40     }  
41     public Mine(int x, int y) {
```

```
42     }  
43     public boolean processAction(ArrayList<ArrayList<Case>> cases) {
```

```
44     }
```

```
45 }
```

```
46 package ds4eti2018_1S.modele;
47 import java.awt.Color;
48 import java.util.ArrayList;
49 import java.util.Iterator;
50
51 public class Demine
52 {
53     public Demine(){
54
55     }
56     public Demine(int x, int y) {
57
58     }
59     @Override
60     public boolean processAction(ArrayList<ArrayList<Case>> cases) {
61
62     }
63
64     }
65
66 }
```

```
62 package ds4eti2018_1S.modele;
```

```
63 import java.awt.Color;
```

```
64 import java.util.ArrayList;
```

```
65  
66 public class Info
```

```
67 {
```

```
68     private int nbMines = 0;
```

```
69     public Info(){
```

```
70  
71  
72  
73
```

```
71     }
```

```
72  
73     public Info(int x, int y) {
```

```
74  
75  
76  
77
```

```
74     }
```

```
75  
76  
77     public boolean processAction(ArrayList<ArrayList<Case>> cases) {
```

```
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100
```

```
78     }
```

```
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100
```

```
79 }
```

```

80 package ds4eti2018_1S.modele;
81 import java.util.ArrayList;
82 import java.util.Iterator;
83 import java.util.Map;
84 import java.util.Map.Entry;
85
86 public class Plateau {
87     public static final int MIN_COL = 5;
88     public static final int MIN_LIGNE = 5;
89     public static final int MAX_COL = 20;
90     public static final int MAX_LIGNE = 20;
91     public static final int DEFAULT_COL = 8;
92     public static final int DEAUPT_LIGNE = 8;
93     public static int longueur=8;
94     public static int largeur=8;
95     private ArrayList<ArrayList<Case>> cases = new ArrayList<ArrayList<Case>>();
96
97     public Plateau(){init();}
98     public Plateau(int x, int y){
99         Plateau.longueur = x;
100        Plateau.largeur = y;
101        init();
102    }
103    public ArrayList<ArrayList<Case>> getCases() {return cases;}
104
105    private void init() {
106        for (int i=0;i<Plateau.longueur;i++){
107            ArrayList<Case> al = new ArrayList<Case>();
108            for(int j=0;j<Plateau.largeur;j++){
109                al.add(tirage(i,j));
110            }
111            cases.add(al);
112        }
113    }
114    private Case tirage(int x, int y){
115        Case ret = null;
116        int alea = (int)(Math.random()*100);
117        boolean trouve = false;
118        Iterator<Entry<Case, Integer>> it = Jeu.tirageAleatoire.entrySet().iterator();
119        while (!trouve && it.hasNext()) {
120            Map.Entry<Case, Integer> entry = (Map.Entry<Case, Integer>)it.next();
121            if(entry.getValue().intValue()>alea){
122                ret = (Case) entry.getKey().clone();
123                ret.setCase(x, y);
124                trouve = true;
125            }
126        }
127        return ret;
128    }
129    public String toString(){

```

```

130 }
131     public static void main(String [] ar){System.out.println(new Plateau(5,8));}
132 }

```

```

133 package ds4eti2018_1S.modele;
134 import java.util.ArrayList;
135 import java.util.LinkedHashMap;
136 import java.util.Map;
137
138 public class Jeu {
139     public static final Map<Case, Integer> tirageAleatoire = createMap();
140     public static final int MAX_JOUEURS = 4;
141     private Plateau plateau;
142     private int nbJoueurs = 1;
143     private int joueur = 1;
144
145     public Jeu(int x, int y, int nbJoueurs){
146
147     }
148     public Jeu(int x, int y){
149
150     }
151     public Jeu(int nbJoueurs){
152
153     }
154     public Jeu(){
155
156     }
157     public ArrayList<ArrayList<Case>> getCases() {return plateau.getCases();}
158     public int getJoueur() {return joueur;}
159
160     private static Map<Case, Integer> createMap()
161     {
162         Map<Case,Integer> myMap = new LinkedHashMap<Case,Integer>();
163         myMap.put(new Demine(), 5);
164         myMap.put(new Mine(), 20);
165         myMap.put(new Info(), 100);
166         return myMap;
167     }
168     public Plateau getPlateau(){return plateau;}
169     public void nextJoueur(){
170
171     }
172     public void rejouer(){plateau=new Plateau();}
173 }

```

```

169 package ds4eti2018_1S.ihm;
170 import java.awt.BorderLayout;
171 import java.awt.Component;
172 import java.awt.Container;
173 import java.awt.Dimension;
174 import java.awt.GridLayout;
175 import javax.swing.JButton;
176 import javax.swing.JComboBox;
177 import javax.swing.JFrame;
178 import javax.swing.JLabel;
179 import javax.swing.JOptionPane;
180 import javax.swing.JPanel;
181 import java.awt.event.ActionEvent;
182 import java.awt.event.ActionListener;
183 import java.util.ArrayList;
184
185 import ds4eti2018_1S.modele.Case;
186 import ds4eti2018_1S.modele.Jeu;
187 import ds4eti2018_1S.modele.Plateau;
188
189 public class Fenetre extends JFrame {
190     private static final int CASE_SIZE = 50;
191     private Jeu jeu;
192     private ArrayList<ArrayList<Case>> cases;
193
194     private JComboBox<Integer> jcbNb;
195     private JComboBox<Integer> jcbCol;
196     private JComboBox<Integer> jcbLigne;
197
198     private JLabel lInfo = new JLabel("C'est au tour du joueur 1");
199     private Container grille = new JPanel();
200     private boolean perdu = false;
201
202     public Fenetre(){
203         super("Mon super démineur");
204         initDialogue();
205     }
206     private void init() {
207         int nbJoueurs = jcbNb.getItemAt(jcbNb.getSelectedIndex()).intValue();
208         int nbCol = jcbCol.getItemAt(jcbCol.getSelectedIndex()).intValue();
209         int nbLigne = jcbLigne.getItemAt(jcbLigne.getSelectedIndex()).intValue();
210
211         jeu = new Jeu(nbLigne,nbCol, nbJoueurs);
212         cases = jeu.getCases();
213         getContentPane().removeAll();
214         grille.removeAll();
215         grille.setLayout(new GridLayout(nbLigne,nbCol));
216

```

```

        for(ArrayList<Case> ligne:cases){

```

```

        }

```

```
217     Container cont = getContentPane();
218     cont.setLayout(new BorderLayout());
219     cont.add(grille, BorderLayout.NORTH);
220     cont.add(lInfo, BorderLayout.SOUTH);
221     pack();
222 }
223 private void initDialogue() {
```

```
224 }
225
```

```

226 private void refresh(){
227     int i=0;
228     boolean gagne = true;
229     for(ArrayList<Case> ligne:cases){
230         for(Case c:ligne){
231             Component bt = grille.getComponent(i);
232             i++;
233             if(c.isVisible()){
234                 bt.setEnabled(false);
235                 bt.setBackground(c.getCouleur());
236                 ((JButton)(bt)).setText(c.toString());
237             }
238             else{gagne=false;}
239         }
240     }
241     if(perdu){
242         JOptionPane.showMessageDialog(this,"le joueur "+jeu.getJoueur()+"a perdu");
243         initDialogue();
244     }
245     else if(gagne){
246         JOptionPane.showMessageDialog(this,"le joueur "+jeu.getJoueur()+"a gagné");
247         initDialogue();
248     }
249     else{
250         jeu.nextJoueur();
251         lInfo.setText("C'est au tour du joueur "+jeu.getJoueur());
252     }
253 }
254 private class ListenerBt implements ActionListener {
255
256     private Case c;
257     public ListenerBt(Case c) {
258         this.c = c;
259     }
260
261 }
262 }

```

## 8 Javadoc et documentation

## 8.1 Point

```
public class Point
extends Point2D
implements Serializable
```

A point representing a location in (x,y) coordinate space, specified in integer precision.

**Since:**

1.0

**See Also:**

[Serialized Form](#)

### ***Nested Class Summary***

#### **Nested classes/interfaces inherited from class java.awt.geom.Point2D**

[Point2D.Double](#), [Point2D.Float](#)

### ***Field Summary***

#### **Fields**

Modifier and Type	Field and Description
int	<b>x</b> The X coordinate of this Point.
int	<b>y</b> The Y coordinate of this Point.

### ***Constructor Summary***

#### **Constructors**

##### **Constructor and Description**

###### **Point()**

Constructs and initializes a point at the origin (0, 0) of the coordinate space.

###### **Point(int x, int y)**

Constructs and initializes a point at the specified (x,y) location in the

## Field Detail

**x**

```
public int x
```

The X coordinate of this Point. If no X coordinate is set it will default to 0.

**Since:**

1.0

**See Also:**

`getLocation()`, `move(int, int)`

**y**

```
public int y
```

The Y coordinate of this Point. If no Y coordinate is set it will default to 0.

**Since:**

1.0

**See Also:**

`getLocation()`, `move(int, int)`

## 8.2 JLabel

### **public void setLabelFor([Component](#) c)**

Set the component this is labelling. Can be null if this does not label a Component. If the `displayedMnemonic` property is set and the `labelFor` property is also set, the label will call the `requestFocus` method of the component specified by the `labelFor` property when the mnemonic is activated.

**Parameters:**

c - the Component this label is for, or null if the label is not the label for a component

### **public void setDisplayedMnemonic(char aChar)**

Specify a keycode that indicates a mnemonic key. This property is used when the label is part of a larger component. If the `labelFor` property of the label is not null, the label will call the `requestFocus` method of the component specified by the `labelFor` property when the mnemonic is activated.

**Parameters:**

aChar - a char specifying the mnemonic to display

## 8.3 JButton

### **public void setMnemonic(char mnemonic)**

This method is now obsolete, please use `setMnemonic(int)` to set the mnemonic for a button. This method is only designed to handle character values which fall between 'a' and 'z' or 'A' and 'Z'.

**Parameters:**

mnemonic - a char specifying the mnemonic value

265        **public void setEnabled(boolean b)**

266        Enables (or disables) the button.

#### 8.4 JComboBox<sup>1</sup>

- Creating a default, empty `JComboBox` then add items later using the `addItem()` method:

```
// create an empty combo box with items of type String
JComboBox<String> comboLanguage = new JComboBox<String>();

// add items to the combo box
comboLanguage.addItem("English");
comboLanguage.addItem("French");
comboLanguage.addItem("Spanish");
comboLanguage.addItem("Japanese");
comboLanguage.addItem("Chinese");
```

- Getting selected item (using the `getSelectedItem()` or `getSelectedIndex()` methods):

```
// get the selected item as an object
String selectedBook = (String) bookList.getSelectedItem();
Job selectedJob = (Job) jobList.getSelectedItem();
// get the selected item as an index:
int selectedIndex = jobList.getSelectedIndex();
```

---

18    1 d'après <http://www.codejava.net/java-se/swing/jcombobox-basic-tutorial-and-examples>