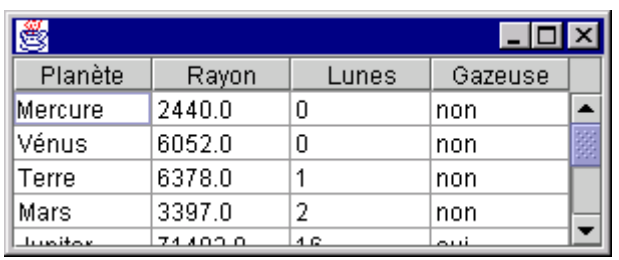
Tables

- Construire une table
- Modèle de table
- Filtrer un modèle

Tables


TestTablePlanetes.bat

- **`JTable`** affiche des données dans un tableau
- **`TableModel`** régit la gestion des données
- On peut fournir les données dans un tableau bidimensionnel d'objets : **`Object[][]`** et utiliser le **`DefaultTableModel`**, mais il vaut mieux étendre **`AbstractTableModel`**.
- La sélection est régi par une ***modèle de sélection***
- De plus, il y a un ***modèle de colonnes***.
- Un tableau est entouré d'ascenseurs, en général.



Construire une table

- Les constructeurs sont

`JTable()` modèles par défaut pour les trois modèles

`JTable(int numRows, int numColumns)` avec autant de cellules vides

`JTable(Object[][] rowData, Object[] columnNames)` avec les valeurs des cellules de **`rowData`** et noms de colonnes **`columnNames`**.

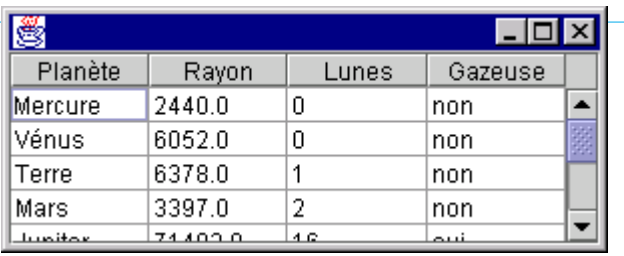
`JTable(TableModel dm)` avec le modèle de données **`dm`**, les autres par défaut.

`JTable(TableModel dm, TableColumnModel cm)` avec modèle de données et modèle de colonnes fournis.

`JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm)` Les trois modèles sont fournis.

`JTable(Vector rowData, Vector columnNames)` ici, les données sont fournies par colonne.

Exemple



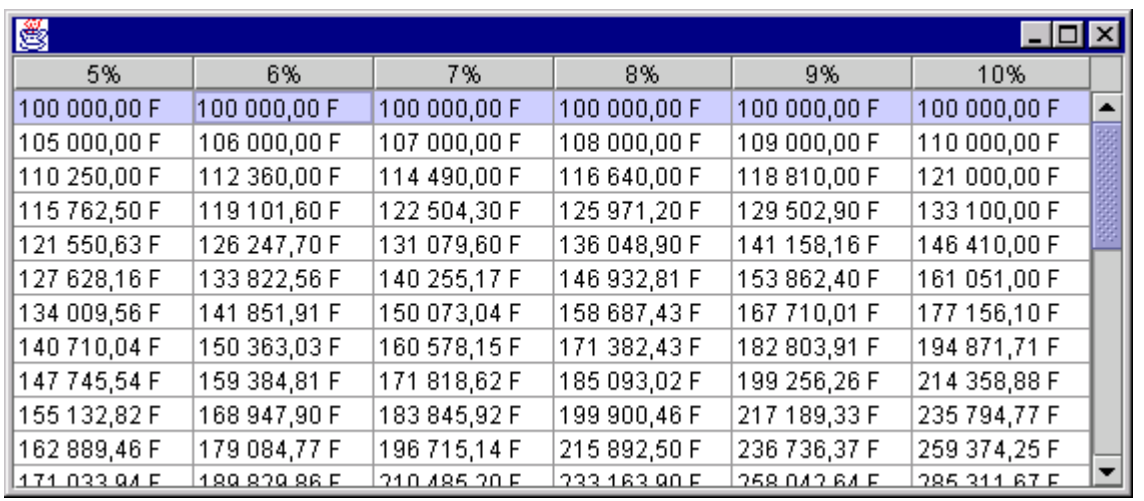

```
class TablePlanetes extends JPanel {
  TablePlanetes() {
    setLayout(new BorderLayout());
    JTable table = new JTable(cellules, columnNames);
    add(new JScrollPane(table), BorderLayout.CENTER);
  }
  private Object[][] cellules = {
    { "Mercure", new Double(2440), new Integer(0), "non"},
    { "Vénus", new Double(6052), new Integer(0), "non"},
    { "Terre", new Double(6378), new Integer(1), "non"},
    { "Mars", new Double(3397), new Integer(2), "non"},
    { "Jupiter", new Double(71492), new Integer(16), "oui"},
    { "Saturne", new Double(60268), new Integer(18), "oui"},
    { "Uranus", new Double(25559), new Integer(17), "oui"},
    { "Neptune", new Double(24766), new Integer(8), "oui"},
    { "Pluton", new Double(1137), new Integer(1), "non"}
  };
  private String[] columnNames =
    {  "Planète", "Rayon", "Lunes", "Gazeuse"};
}
```

Modèles de table

- Les données sont accessible par un ***modèle***. Ils peuvent être stockés ou calculés, de façon transparente.
- La classe **`AbstractTableModel`** implémente les méthodes d'un modèle de table, sauf

```
public int getRowCount()
public int getColumnCount()
public Object getValueAt(int ligne, int colonne)
```

- qui retournent respectivement
  - le nombre de lignes
  - le nombre de colonnes
  - l'objet à afficher dans les ligne et colonne indiquées (sa méthode **`toString`** est utilisée).

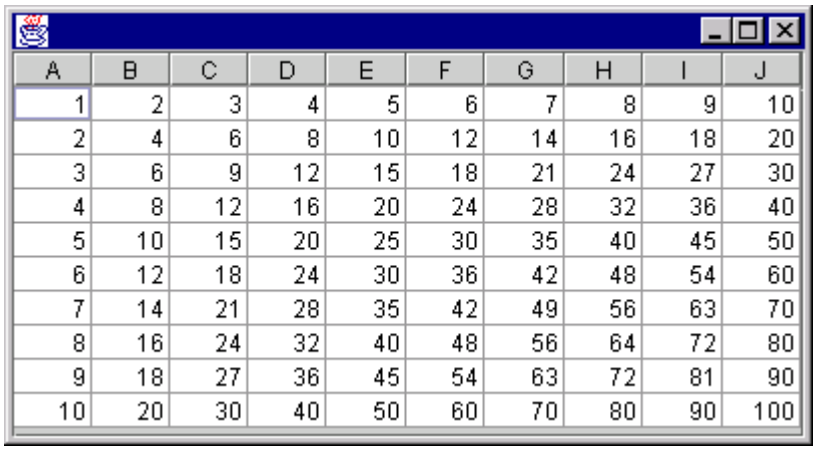


5%	6%	7%	8%	9%	10%
100 000,00 F	100 000,00 F	100 000,00 F	100 000,00 F	100 000,00 F	100 000,00 F
105 000,00 F	106 000,00 F	107 000,00 F	108 000,00 F	109 000,00 F	110 000,00 F
110 250,00 F	112 360,00 F	114 490,00 F	116 640,00 F	118 810,00 F	121 000,00 F
115 762,50 F	119 101,60 F	122 504,30 F	125 971,20 F	129 502,90 F	133 100,00 F
121 550,63 F	126 247,70 F	131 079,60 F	136 048,90 F	141 158,16 F	146 410,00 F
127 628,16 F	133 822,56 F	140 255,17 F	146 932,81 F	153 862,40 F	161 051,00 F
134 009,56 F	141 851,91 F	150 073,04 F	158 687,43 F	167 710,01 F	177 156,10 F
140 710,04 F	150 363,03 F	160 578,15 F	171 382,43 F	182 803,91 F	194 871,71 F
147 745,54 F	159 384,81 F	171 818,62 F	185 093,02 F	199 256,26 F	214 358,88 F
155 132,82 F	168 947,90 F	183 845,92 F	199 900,46 F	217 189,33 F	235 794,77 F
162 889,46 F	179 084,77 F	196 715,14 F	215 892,50 F	236 736,37 F	259 374,25 F
171 033,94 F	189 829,86 F	210 485,20 F	233 163,90 F	258 042,64 F	285 311,67 F

Un premier exemple


TrivTable.bat

- En plus des trois méthodes obligées, la fonction **getColumClass** a été redéfinie, ce qui produit la justification à droite



A	B	C	D	E	F	G	H	I	J
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

```
class SimpleTable extends JPanel {
  SimpleTable() {
    setLayout(new BorderLayout());
    TableModel dataModel = new AbstractTableModel() {
      public int getColumnCount() { return 10; }
      public int getRowCount() { return 10;}
      public Object getValueAt(int row, int col) {
        return new Integer((1+row)*(1+col));
      }
      public Class getColumnClass(int column) {
        return Number.class;
      }
    };
    JTable table = new JTable(dataModel);
    add(new JScrollPane(table));
  }
}
```

Un deuxième exemple

TestTableInvestissements.bat

5%	6%	7%	8%	
100 000,00 F	100 000,00 F	100 000,00 F	100 000,00 F	1
105 000,00 F	106 000,00 F	107 000,00 F	108 000,00 F	1
110 250,00 F	112 360,00 F	114 490,00 F	116 640,00 F	1
115 762,50 F	119 101,60 F	122 504,30 F	125 971,20 F	1
121 550,63 F	126 247,70 F	131 079,60 F	136 048,90 F	1
127 628,16 F	133 822,56 F	140 255,17 F	146 932,81 F	1
134 009,56 F	141 851,91 F	150 073,04 F	158 687,43 F	1

- Construction par

```
TableModel model = new ModelInvestment(30, 5, 10);
JTable table = new JTable(model);
```

- avec bien entendu

```
class ModelInvestment extends AbstractTableModel {...}
```

- méthodes à écrire (plus **`getColumnName`** qui, par défaut, numérote **`A, B,`** etc):

```
public int getRowCount()
public int getColumnCount()
public Object getValueAt(int ligne, int colonne)

public String getColumnName(int colonne)
```

Détails

```
class ModelInvestment extends AbstractTableModel {
  private int annees;
  private int tauxMin;
  private int tauxMax;
  private static double depot = 100000.0;

  ModelInvestment(int annees, int tauxMin, int tauxMax) {
    this.annees = annees;
       this.tauxMin = tauxMin;
       this.tauxMax = tauxMax;
   }

  public int getRowCount() { return annees;}

  public int getColumnCount() { return tauxMax - tauxMin + 1;}

  public Object getValueAt(int ligne, int colonne) {
    double taux = (colonne + tauxMin) / 100.0;
    double depotFinal = depot * Math.pow(1 + taux, ligne);
    return NumberFormat.getCurrencyInstance().format(depotFinal);
   }

  public String getColumnName(int colonne) {
    double taux = (colonne + tauxMin) / 100.0;
    return NumberFormat.getPercentInstance().format(taux);
   }
}
```

Format de nombres

- La classe abstraite **`java.text.NumberFormat`** est la classe de base pour le formatage de nombres.
- Nombreuses méthodes statiques retournant des formats appropriés.
- Le formatage effectif se fait par la méthode du format

```
String format(int donnée)
```

- Exemples de méthodes:

```
NumberFormat.getNumberInstance()
NumberFormat.getCurrencyInstance()
NumberFormat.getPercentInstance()
```

- le format dépend de la **`Locale`**, c'est-à-dire du pays concerné.

Autres usages

- La méthode

```
boolean TableModel.isCellEditable(int l, int c)
```

renvoie true si la cellule peut être modifiée (par défaut non)

- La méthode

```
int JTable.columnAtPoint(Point p)
```

renvoie l'indice de la colonne du tableau où est le point

- La méthode

```
int JTable.convertColumnIndexToModel(int colonne)
```

renvoie l'indice, dans le modèle, de l'indice **`colonne`**

Exemple : tri par ligne


TestTriTable.bat

- On veut trier les lignes, en fonction d'une colonne désignée par un double clic.
- On veut que le tri se fasse sur la vue, pas sur le modèle, pour que le modèle soit inchangé.

Planète	Rayon	Lunes	Gazeuse
Pluton	1137.0	1	non
Mercure	2440.0	0	non
Mars	3397.0	2	non
Vénus	6052.0	0	non
Terre	6378.0	1	non
Neptune	24766.0	8	oui
Uranus	25559.0	17	oui
Saturne	60268.0	18	oui
Jupiter	71492.0	16	oui

- Pour cela, on introduit un ***filtre de modèle***, similaire aux filtres de streams.
- Ce filtre enregistre une référence au modèle réel, et intercepte les communications entre la table et son modèle pour les réinterpréter.
- Le filtre maintient une permutation des lignes déterminée par le ***tri virtuel*** des lignes en fonction de la colonne choisie.

Usage

- Le modèle est associé à la table

```
class TablePlanetes extends JPanel {
  TablePlanetes() {
    setLayout(new BorderLayout());
    DefaultTableModel model = new DefaultTableModel(cellules, columnNames);
    FiltreTriModel sorter = new FiltreTriModel(model);
    JTable table = new JTable(sorter);
    sorter.addEcouteur(table);
    add(new JScrollPane(table), BorderLayout.CENTER);
  }
  private Object[][] cellules = { ... };
  private String[] columnNames = { ... };
}
```

Tri

- La classe **`FiltreTriModel`** est un modèle de table avec
  - une référence au modèle réel
  - un tableau de lignes
  - une méthode de tri de ce tableau

```
class FiltreTriModel extends AbstractTableModel {
  public FiltreTriModel(TableModel m) {
    model = m;
    lignes = new Ligne[model.getRowCount()];
    for (int i = 0; i < lignes.length; i++) {
      lignes[i] = new Ligne();
      lignes[i].index = i;
    }
  }

  public void sort(int c) {
    colonneTri = c;
    Arrays.sort(lignes);
    fireTableDataChanged();
  }
...
  private TableModel model;
  private int colonneTri;
  private Ligne[] lignes;
}
```

- Le tri est fait par la méthode statique de la classe **`Arrays`**, en fonction de la colonne choisie.
- Pour cela, la classe **`Ligne`** doit implémenter l'interface **`Comparable`**.

Tri (suite)

- La classe **`Ligne`** est interne à **`FiltreTriModel`** pour accéder facilement aux données.
- Une ligne est plus petite qu'une autre si son élément dans la colonne de tri est plus petit que l'élément de cette même colonne dans l'autre ligne.

```
class FiltreTriModel extends AbstractTableModel {
...
  private class Ligne implements Comparable {
    public int index;
    public int compareTo(Object autre) {
      Ligne autreLigne = (Ligne)autre;
      Object cellule = model.getValueAt(index, colonneTri);
      Object autreCellule = model.getValueAt(autreLigne.index, colonneTri);
      return ((Comparable)cellule).compareTo(autreCellule );
    }
  }
}
```

Filtre

- Les trois fonctions obligées tiennent compte de l'ordre virtuel

```
class FiltreTriModel extends AbstractTableModel {
...
  public Object getValueAt(int r, int c) {
    return model.getValueAt(lignes[r].index, c);
  }

  public int getRowCount(){  return model.getRowCount();}
  public int getColumnCount(){  return model.getColumnCount();}
  public String getColumnName(int c){  return model.getColumnName(c);}
  public Class getColumnClass(int c){
    return (c == 1 || c == 2) ? Number.class : Object.class;
  }
}
```

Et le double clic

- C’est l’en-tête de colonne qui réagit.
- Au double clic sur une colonne, on récupère le numéro de la colonne dans le modèle

```
class FiltreTriModel extends AbstractTableModel {
...
  public void addEcouteur(final JTable table) {
    table.getTableHeader().addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent event) {
        if (event.getClickCount() < 2) return;
        int tableColumn = table.columnAtPoint(event.getPoint());
        int modelColumn = table.convertColumnIndexToModel(tableColumn);
        sort(modelColumn);
      }
    });
  }
}
```

Evénements

- La classe **`JTable`** écoute les événements reçus du modèle
- Le modèle a plusieurs méthodes pour signaler des modifications de données

```
fireTableDataChanged()
fireTableStructureChanged()
fireTableRowsInverted(int first, int last)
fireTableRowsUpdated(int first, int last)
fireTableRowsDeleted(int first, int last)
fireTableCellUpdated(int row, int col)
fireTableChangedEvent(TableModelEvent e)
```

- Les colonnes sont régies par un **`TableColumnModel`** qui a ses propres notificateurs d'événements