

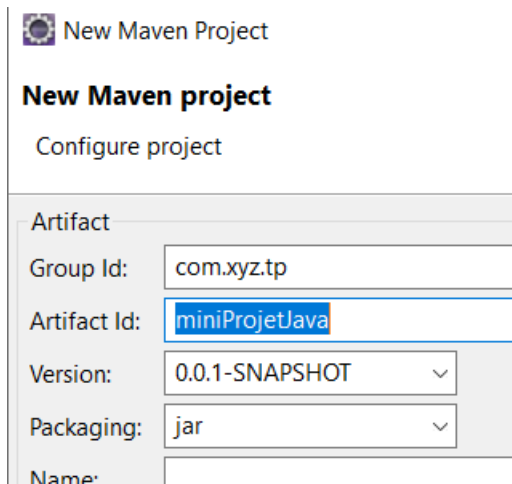
1. Mini projet java (révisions)

Objectif :

Programmer une petite application java qui générera un fichier "dessin.svg" affichable via un navigateur internet .

1.1. création et paramétrage du projet

Créer (avec eclipse) un nouveau projet "maven" simple intitulé "*miniProjetJava*"



New Maven Project

New Maven project

Configure project

Artifact

Group Id: com.xyz.tp

Artifact Id: miniProjetJava

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Seuls paramètres importants (indispensables) à ajouter dans **pom.xml** :

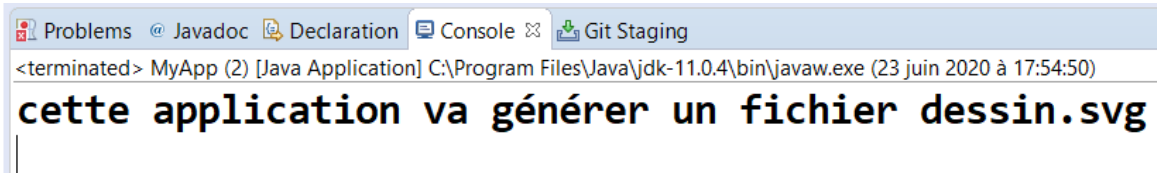
```
<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

NB : penser à activer le menu contextuel "**maven / update project ...**" (en se plaçant à la racine du projet dans le "project explorer") .

Effet escompté : JavaSE-1.5 --> **JavaSE-1.8**

1.2. classe principale tp.MyApp

- générer un nouveau package "**tp**" dans src/main/java
- générer une nouvelle classe "**MyApp**" contenant la méthode principale **main()** dans le package "**tp**".
- Au sein de la méthode main() afficher à la console le message **"cette application va générer un fichier dessin.svg"**
- Lancer une première fois l'exécution de l'application java et vérifier le bon fonctionnement au sein de la console :



The screenshot shows the Eclipse IDE's console window. The title bar includes tabs for Problems, Javadoc, Declaration, Console, and Git Staging. The console output shows the command prompt for the 'MyApp (2) [Java Application]' project, indicating the Java version is 11.0.4 and the execution time is 23 juin 2020 à 17:54:50. The output text is: **cette application va générer un fichier dessin.svg**

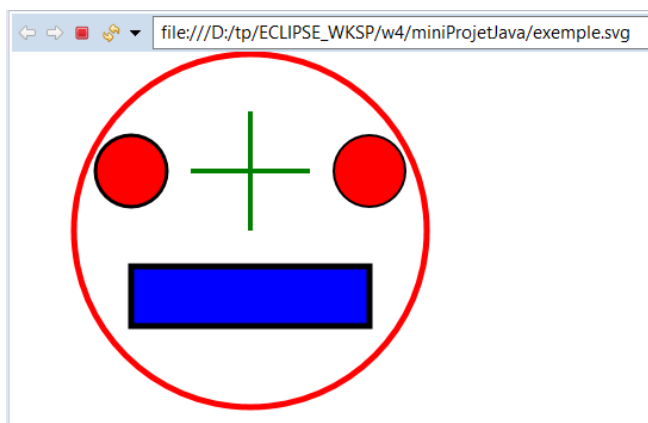
1.3. exemple de fichier ".svg"

créer à coté de pom.xml un **nouveau fichier** vide **exemple.svg**

via click droit / **open with / text editor ...** placer (via un copier/coller) le contenu suivant :

```
<svg xmlns='http://www.w3.org/2000/svg' height='400' width='500'>
  <circle cx='100' cy='100' r='30' stroke='black' stroke-width='3' fill='red' />
  <line x1='200' y1='50' x2='200' y2='150' stroke='green' stroke-width='4' fill='none' />
  <line x1='150' y1='100' x2='250' y2='100' stroke='green' stroke-width='4' fill='none' />
  <circle cx='300' cy='100' r='30' stroke='black' stroke-width='2' fill='red' />
  <rect x='100' y='180' width='200' height='50'
    fill='blue' stroke='black' stroke-width='5' />
  <circle cx='200' cy='150' r='148' stroke='red' stroke-width='5' fill='none' />
</svg>
```

via click droit / **open with / internal web browser** , visualiser ce dessin vectoriel au sein du navigateur web intégré à l'ide eclipse



1.4. partie abstraite de tp.figure

créer (sous tp) le sous **package tp.figure**

Au sein du package tp.figure , coder les 2 interfaces suivantes :

```
public interface Surface {
    public double perimetre();
    public double aire();
}
```

et

```
public interface Transformable {
    public void translater(int dx,int dy);
    public void zoomer(double coeff);
}
```

créer (dans tp.figure) une *nouvelle classe abstraite* "Figure2D" avec le code initial suivant :

```
package tp.figure;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

public abstract class Figure2D implements Transformable{
    private String couleur="black"; //couleur du trait ou du contour ("black" par défaut)
    private Integer epaisseur=1; //épaisseur du trait ou du contour (1 par défaut)
    private String couleurFond; //null par défaut ("none" en svg)

    //design pattern "template method" avec polymorphisme sur sous tâche abstraite .
    public String toSvgStringWithColor() {
        //polymorphisme sur l'appel à this.toSvgSubString();
        //où this pourra référencer une instance de Cercle ou Ligne ou Rectangle
        String beginOfSvgString = this.toSvgSubString();

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        PrintStream ps = new PrintStream(baos);
        ps.printf("%s stroke='%s' stroke-width='%d' fill='%s'/>",
            beginOfSvgString ,
            this.couleur ,
            this.epaisseur ,
            this.couleurFond==null?"none":this.couleurFond);

        return baos.toString();
    }
    public abstract String toSvgSubString();
    //+get/set , +constructeurs , +toString()
}
```

Compléter ce code en générant via les assistants d'eclipse les éléments habituels
(+get/set , +constructeurs , +toString())

1.5. partie concrète de tp.figure

Créer au sein de tp.figure , 3 nouvelles classes concrètes :

- **Ligne** héritant de Figure2D
- **Cercle** héritant de Figure2D et implémentant l'interface Surface .
- **Rectangle** héritant de Figure2D et implémentant l'interface Surface .

NB :

- Ces 3 classes devront également coder les méthodes imposées par l'interface Transformable (indirectement récupérée par l'héritage de Figure2D déclarant implémenter l'interface Transformable) .
- Chacune de ces classes devra comporter au minimum 2 constructeurs :
 - un constructeur par défaut (avec zéro argument)
 - un constructeur avec toutes les coordonnées comme premiers arguments importants et avec les couleur, epaisseur et couleurFond comme derniers arguments secondaires

Coordonnées internes d'une ligne :

```
private int x1;  
private int y1;  
private int x2;  
private int y2;
```

Coordonnées internes d'un cercle :

```
private int cx;//x du centre du cercle  
private int cy;//y du centre du cercle  
private int r; //rayon
```

Coordonnées internes d'un rectangle :

```
private int x;  
private int y;  
private int width;  
private int height;
```

NB : On pourra coder les méthodes imposées petit à petit en les testant au fur et à mesure de leur mise en oeuvre.

Autrement dit , certaine méthodes pourront temporairement être laissées avec le code par défaut généré par les assistants d'eclipse .

Les méthodes délicates `toSvgSubString()` seront codées et testées ultérieurement .

1.6. premiers tests via la classe tp.MyApp

Coder au sein de tp.MyApp quelques tests de ce genre :

```
Rectangle r = new Rectangle(100,180,200,50,"black",5,"blue");
System.out.println(r.toString());
System.out.println("r.perimetre="+r.perimetre());
System.out.println("r.aire="+r.aire());
```

//idem pour Cercle

//idem pour Ligne (sans périmetre ni aire)

1.7. coder et tester les méthodes `toSvgSubString()`

La méthode `.toSvgSubString()` de la classe **Ligne** devra générer et retourner une chaîne de caractère de ce type : `<line x1='200' y1='50' x2='200' y2='150'`

La méthode `.toSvgSubString()` de la classe **Cercle** devra générer et retourner une chaîne de caractère de ce type : `<circle cx='300' cy='100' r='30'`

La méthode `.toSvgSubString()` de la classe **Rectangle** devra générer et retourner une chaîne de caractère de ce type : `<rect x='100' y='180' width='200' height='50'`

NB : étant donné que la méthode `toSvgStringWithColor()` héritée de la classe abstraite **Figure2D** ajoute une partie finale de de genre `stroke='black' stroke-width='3' fill='red' />`

on testera indirectement la méthode `.toSvgSubString()` en appelant `.toSvgStringWithColor()` sur des lignes , des cercles et des rectangles .

exemple :

```
System.out.println(r.toSvgStringWithColor());
```

1.8. coder et tester `tp.svg.MySvgUtil.generateSvgFile()`

- créer un nouveau package `tp.svg`
- créer et coder la nouvelle classe suivante :

```
package tp.svg;

import java.util.List;

import tp.figure.Figure2D;

public class MySvgUtil {

    public static void generateSvgFile(List<Figure2D> listeFig , String fileName) {
        //ex de fileName : "dessin.svg"

        //ouvrir le fichier en écriture (flux élémentaire + PrintStream)

        //générer/écrire via .println() la première ligne du fichier svg
        //<svg xmlns='http://www.w3.org/2000/svg' height='400' width='500'>

        //boucler sur chaque élément de la liste listeFig
        //et générer/écrire via .println() une ligne dont la valeur est construite
        //via la méthode .toSvgStringWithColor()

        //générer/écrire via .println() la dernière ligne du fichier svg
        //</svg>

        //fermer les flux ouverts
    }
}
```

Pour tester cette classe , on ajoutera dans la méthode `main()` de `tp.MyApp` du code qui :

- créera une instance de `ArrayList<Figure2D>`
- y ajoutera quelques instances de `Cercle` , `Ligne` et `Rectangle`
- appellera la méthode statique . `generateSvgFile` de `MySvgUtil` .

On pourra enfin :

- vérifier l'existence du fichier généré *dessin.svg* via un **Refresh** eclipse
- visualiser le fichier généré via *open with /text editor* et *open with / internal web browser*

Une éventuelle suite facultative de ce miniProjet pourra être effectuée plus tard (transformations de coordonnées , lambda , stream , ...).