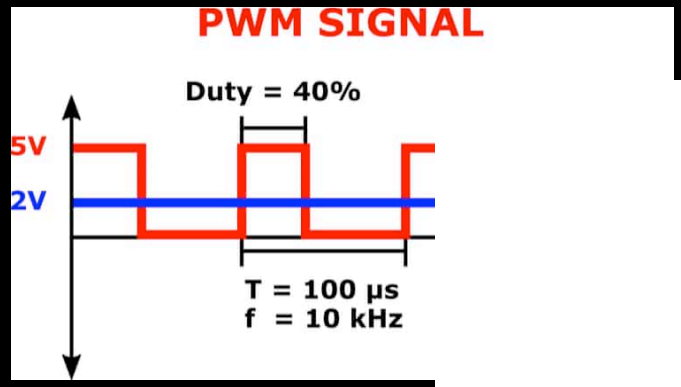
ATELIER PROGRAMMATION

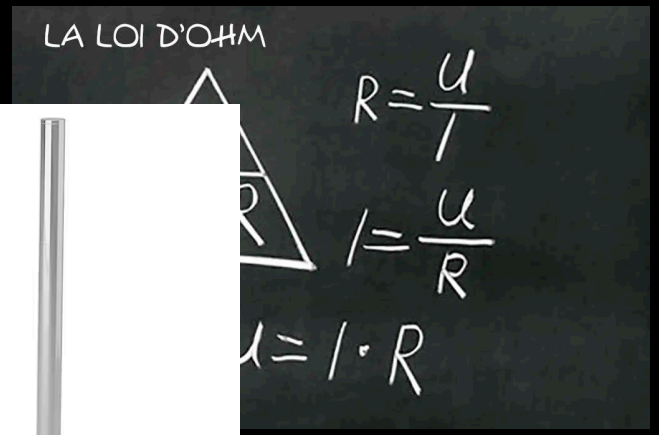
DÉFI HÉRITAGE

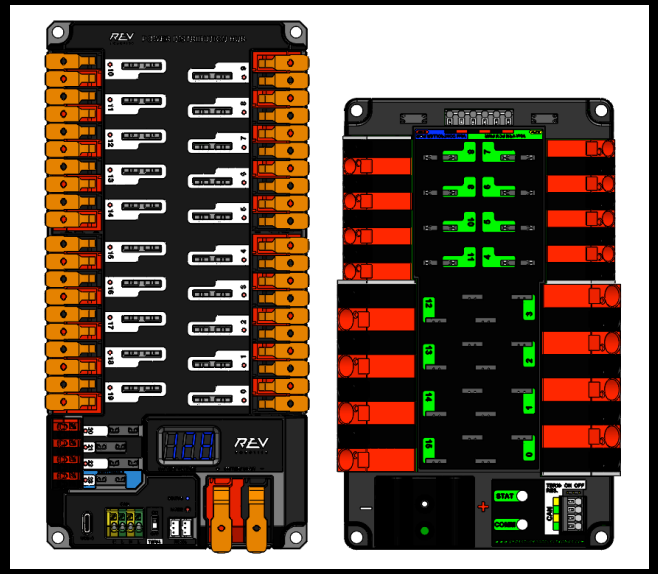
AGENDA

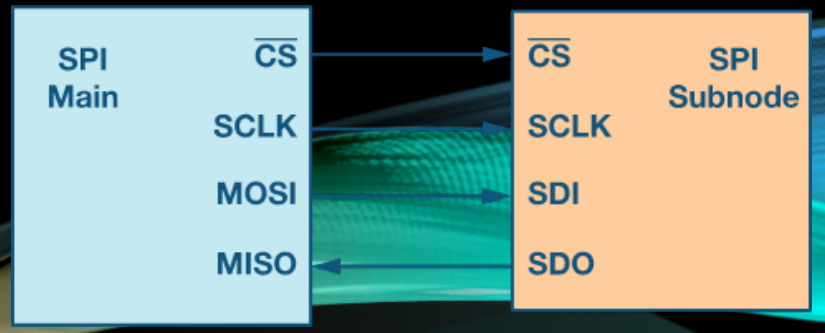
- Retour sur l'atelier d'électronique
- Les équipes et le matériel
- Un peu d'administration
- Git-graph
- Les prochaines semaines
- une intro rapide à UML
- Sans plus attendre… votre défi!

RETOUR SUR L'ATELIER D'ÉLECTRONIQUE



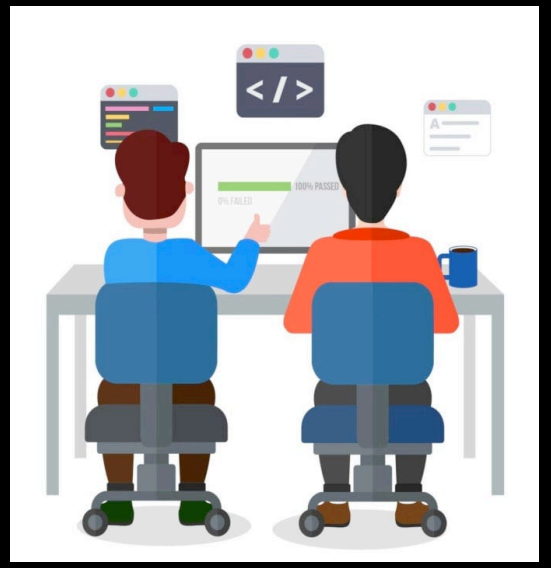






LES ÉQUIPES ET LE MATÉRIEL

Prenez le laptop correspondant à votre # d'équipe



Les équipes:

1. **Héloïse et Noémie**
2. **Victor et Shawn**
3. **Félix et Émeryck**
4. **Victoria et Maelie**
5. **Igor et Caera**
6. **Medhi et Louis-Richard**

Pour le défi, nous allons grouper les équipes 1, 2 et 3 pour former le **groupe A** et 4,5 et 6 pour former le **groupe B**.

UN PEU D'ADMINISTRATION

- Pour conserver vos configurations personnelles, vous allez devoir vous créer un compte sur les laptop Linux.
- Il vous faut un compte par personne (pas par équipe)
- vous ne devriez JAMAIS travailler sous l'utilisateur hyperion mais bien utiliser votre compte

https://linux.how2shout.com/how-to-create-user-accounts-in-linux-mint-graphically/

https://topnewreview.com/creating-users-in-linux-mint/

GIT-GRAPH

Apprendre à installer des extensions dans VsCode:

https://code.visualstudio.com/docs/editor/extension-marketplace

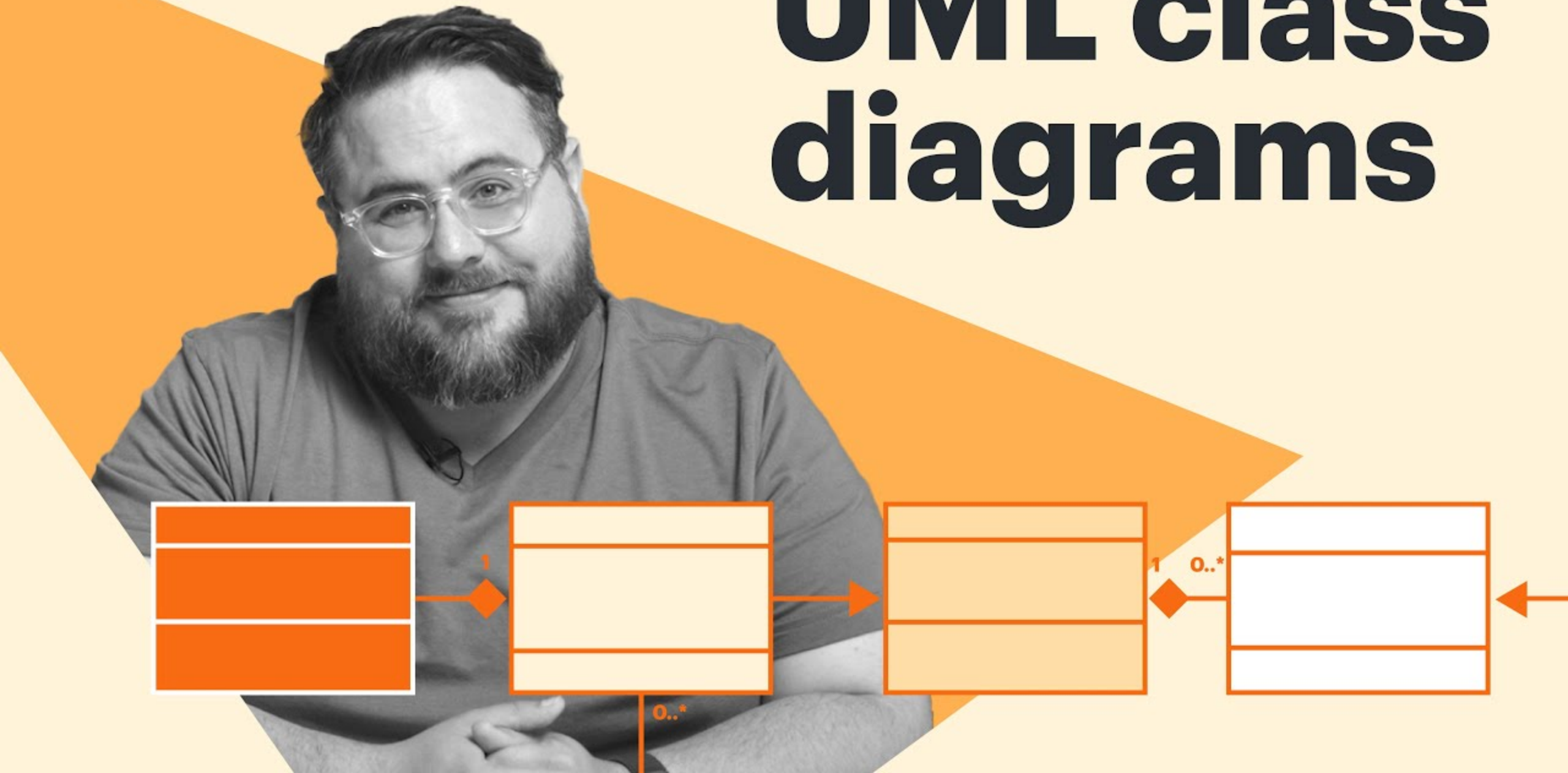
Installez l'extension Git graph dans votre instance de Visual Code:

https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph

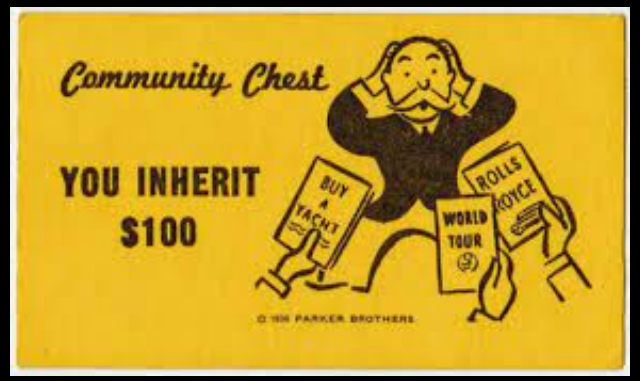
LES PROCHAINES SEMAINES

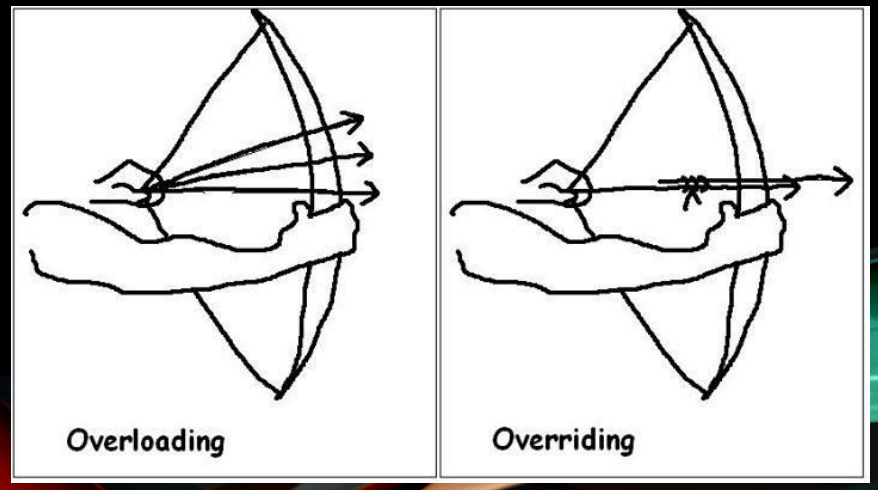
- 8, 13 et 15 Novembre sont pour le défi Shape
- 20 et 22 Novembre sont pour l'introduction à WPIlib
- 27 et 29 Novembre: défi Romi et Victor's testbench
- 4 décembre présentation du code de vendetta par Rosemarie
- 6, 11, 13, 18 et 20 décembre la simulation de compétition par groupe

UML class diagrams



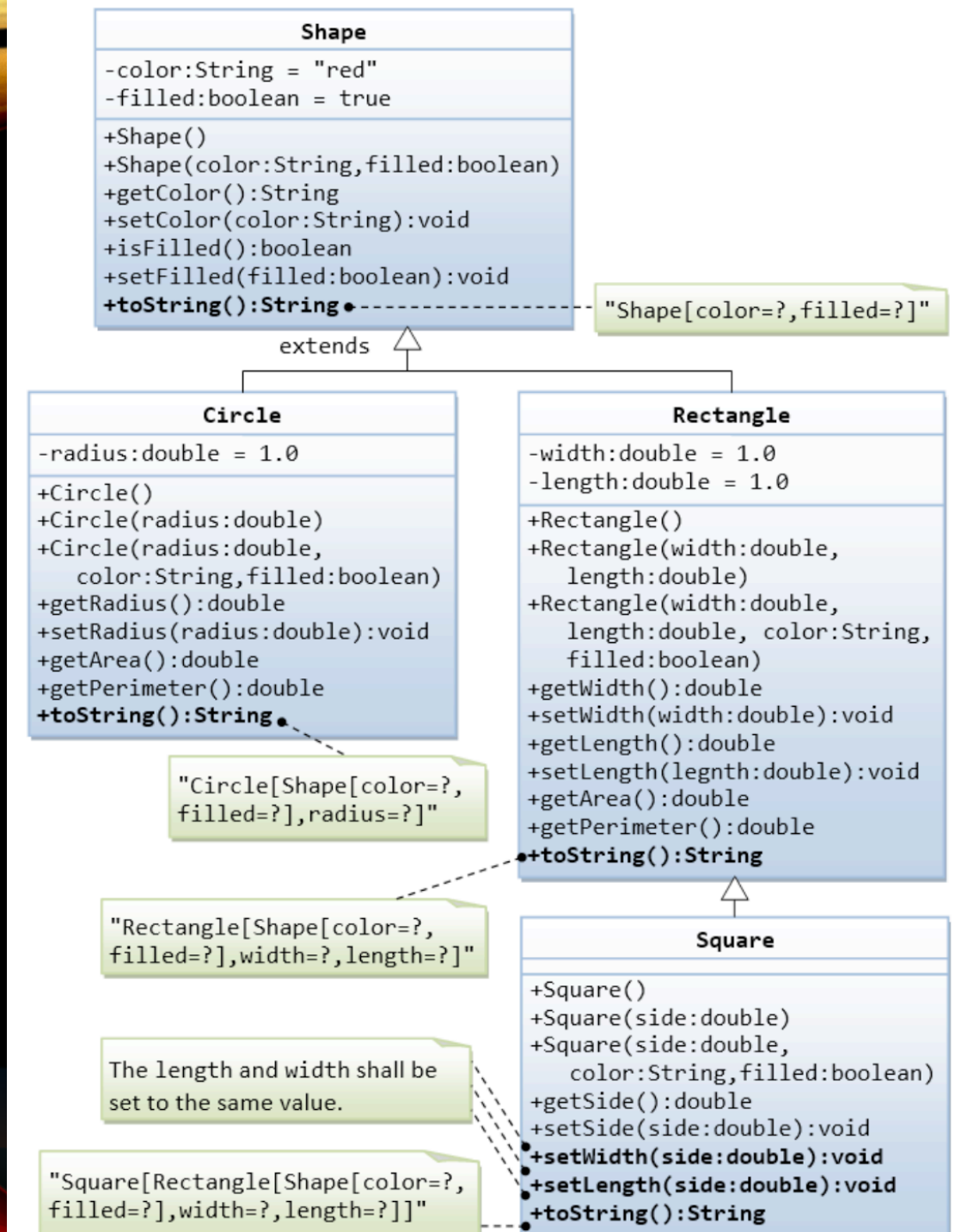
SANS PLUS ATTENDRE… VOTRE DÉFI





UNIFIED MODELING LANGUAGE™

UML®


Shape
-color:String = "red"
-filled:boolean = true
+Shape()
+Shape(color:String,filled:boolean)
+getColor():String
+setColor(color:String):void
+isFilled():boolean
+setFilled(filled:boolean):void
+toString():String
"Shape[color=?,filled=?]"
extends
Circle
-radius:double = 1.0
+Circle()
+Circle(radius:double)
+Circle(radius:double,
color:String,filled:boolean)
+getRadius():double
+setRadius(radius:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Circle[Shape[color=?,
filled=?],radius=?]"
Rectangle
-width:double = 1.0
-length:double = 1.0
+Rectangle()
+Rectangle(width:double,
length:double)
+Rectangle(width:double,
length:double, color:String,
filled:boolean)
+getWidth():double
+setWidth(width:double):void
+getLength():double
+setLength(legnth:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Rectangle[Shape[color=?,
filled=?],width=?,length=?]"
Square
+Square()
+Square(side:double)
+Square(side:double,
color:String,filled:boolean)
+getSide():double
+setSide(side:double):void
+setWidth(side:double):void
+setLength(side:double):void
+toString():String
The length and width shall be
set to the same value.
"Square[Rectangle[Shape[color=?,
filled=?],width=?,length=?]]"

Écrire une classe de base appelée Shape (comme illustré sur le diagramme de classe de la page précédent) qui contient:

- Deux variables de classes privées (ou protégées à vous de voir) nommées *color* de type **String** et *filled* de type **boolean**
- Deux constructeurs: un qui ne prend aucun argument et qui initialise la variable de classe *color* à la valeur **vert** et *filled* à **true**, un autre constructeur qui prend en paramètre des valeurs qui seront assignés à *color* et *filled*.
- Des setter and setter pour toutes les variables de classe (comme illustré dans le diagramme). Par convention, le getter pour une variable de type **boolean** xxx se nommera isXXX() (au lieu de getXxx() pour tous les autres types).
- Une méthode toString qui retourne “Une Shape de couleur xxx et remplie/vide".

Écrire une classe Circle, dérivé de Shape qui contient:

- Une variable d'instance nommée *radius* de type **double** de visibilité private
- Trois constructeurs, tel qu'illustré dans le diagramme de classe (un sans paramètre, un autre qui ne prend que le rayon du cercle et le dernier qui prend le *radius*, la *color* et le *filled*)
- Getter et setter pour la variable de classe *radius*
- Deux méthodes getArea() and getPerimeter() (pour obtenir l'aire et le périmètre du cercle)
- Override la méthode toString() héritée, pour retourner "Un Circle de rayon=xxx, qui est une classe dérivée de yyy", où yyy est la valeur de retour de la méthode toString() de la classe de base.

Écrire une classe Rectangle qui contient:

- Deux variables de classe *width* de type **double** et *length* également de type **double**.
- Trois constructeurs: un sans argument qui initialise width et length à 1.0 et les deux autres tel qu'illustré dans le diagramme de classe
- Getter et setter pour toutes les variables de classe.
- Les méthodes getArea() et getPerimeter() pour obtenir respectivement l'aire et le périmètre du rectangle.
- Override la méthode toString() héritée pour retourner "Un Rectangle de largeur=xxx et de longueur=zzz, qui est une classe dérivée de yyy", où yyy la valeur de retour de la méthode toString() de la classe de base.

Écrire une classe Square, qui est une classe dérivée de Rectangle. Se convaincre que Square peut être conceptualisé comme une classe dérivée de Rectangle. Square n'a pas de variable de classe, mais hérite de *width* et *length* de la classe de base Rectangle.

- Écrivez les constructeurs tels que spécifiés par le diagramme de classe. Indice:

```
public Square(double side)
 {
  super(side, side);  // Call superclass Rectangle(double, double)
 }
```

- Override la méthode toString() pour retourner "A Square avec un côté=xxx, qui est une classe dérivée de yyy", où yyy est la valeur de retour de la méthode toString() de la classe de base de Square.
- Avez-vous besoin d'override getArea() et getPerimeter()?
- Override les méthodes setLength() et setWidth() pour changer les deux variables *width* et *length* de manière à conserver la géométrie d'un carré.

Écrire un programme qui crée un tableau de Shape (statique ou dynamique à votre convenance) et initialisez ce tableau avec 2 formes de chaque type avec des caractéristiques différentes (vous aurez donc 6 formes géométriques au total)

Pour chaque forme, faites afficher à la console les paramètres de l'objet (indice: la méthode toString() pourrait vous être utile surtout implicitement!) et ensuite l'aire et le périmètre de chaque forme géométrique.