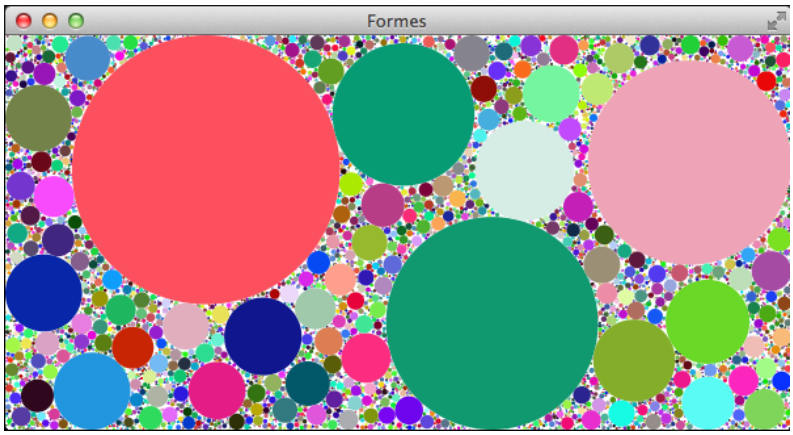
TD11 / TP11

L'objectif de ce TD/TP est de générer des dessins comme ça :



Dessiner des ronds colorés

Vérifions notre capacité à dessiner des ronds colorés partir de l'exemple suivant :

```java
import javafx.application.Application;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.layout.BorderPane;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import java.util.List;
import java.util.ArrayList;
import javafx.scene.shape.Circle;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;

public class DessinExemple extends Application {

    private  int largeur = 600;
    private  int hauteur = 300;

    @Override
    public void start(Stage stage) {
       List<Circle> liste = new ArrayList<>();

    // === Code à modifier ===========

       Circle c = new Circle(150,200,47);
       c.setFill(new Color(Math.random(),Math.random(),Math.random(),1.0));
       liste.add(c);

    // ================================

      Group dessinCercles = new Group();
       dessinCercles.getChildren().addAll(liste);

       BorderPane root = new BorderPane();
       root.setCenter(dessinCercles);
       VBox vbox =this.informations(liste);
       root.setBottom(vbox);
       Scene scene = new Scene(root, this.largeur, this.hauteur+vbox.getHeight()+40);
       stage.setTitle("Formes");
       stage.setScene(scene);
       stage.show();
   }


    private VBox informations(List<Circle> liste){
        VBox vbox = new VBox();
        String cssLayout = "-fx-border-color: black;\n" + "-fx-border-width: 2;\n";
        vbox.setStyle(cssLayout);
        // Décommentez et complétez le code ici ======

        // vbox.getChildren().add(new Label("Le Cercle le plus petit est : "));
        // vbox.getChildren().add(new Label("La surface totale est : "));

        // ==========================================
        return vbox;
    }
    public static void main(String args[]){
       launch(args);
   }
}
```

1. Quel est le profil du constructeur de `Circle` utilisé ? Précisez le rôle de chaque paramètre.
2. Dans quelle classe se trouve le code de la méthode `setFill` utilisée ici ?
3. Le profil de cette méthode est `public final void setFill(Paint value)`. Or lors de l'appel de cette méthode, le paramètre est de type `Color`. Expliquez pourquoi le compilateur ne détecte aucun problème.

4. Quel est le profil du constructeur de `Color` utilisé ici ? Précisez le rôle de chaque paramètre.
5. Modifiez le code pour que le rond coloré soit généré à un emplacement aléatoire dans la fenêtre.
6. Modifiez le code pour générer 100 ronds colorés dans la fenêtre.

Design Objet

Bravo ! Vous avez un dessin avec 100 ronds colorés. Mais il ne ressemble par encore au dessin attendu. En effet, il reste deux problèmes à résoudre :

- on ne veut pas que les ronds se chevauchent ;
- on veut des ronds *les plus grands possible* !

Et tout ça en modifiant le moins possible la classe `Dessin` !

Pour cela, on se propose ici d'utiliser l'algorithme suivant, qui vous surprendra par la *créativité* dont a fait preuve son auteur :

```
Pour i de 0 à 100:
   Créer un nouveau Cercle qui devra choisir son centre dans un 'endroit libre',
   et choisir son rayon de façon à être aussi grand que possible.
```

Pour choisir un 'endroit libre', on utilisera l'*ingénieux* algorithme suivant :

```
Choisir un point au hasard dans le cadre. Si le cercle de rayon 0.1 et centré en ce
point
intersecte un cercle existant ou n'est pas dans le cadre, recommencer.
```

Et pour avoir le cercle *le plus grand possible*, on va commencer par un algorithme simple mais peu efficace, que l'on améliorera plus tard :

```
En partant d'une taille de 0.1, on augmente le rayon du cercle de 0.1 en 0.1 jusqu'à ce
qu'il intersecte une autre cercle ou qu'il ne soit plus dans le cadre.
```

On se propose donc de créer une classe `Cercle` héritant de `Circle` et contenant des méthodes qui vont nous simplifier l'implémentation de l'algorithme ci-dessus.

Pour information, la classe `Circle` contient (entre autres) les méthodes suivantes :

- `public double getRadius()` et `public void setRadius(double r)`,
- `public double getCenterX()` et `public void setCenterX(double x)`,
- `public double getCenterY()` et `public void setCenterY(double y)`,

1. Commencez par définir une classe `Cercle` héritant de `Circle` et contenant un constructeur construisant un cercle de couleur aléatoire, de position aléatoire, et de rayon aléatoire.
2. Modifiez le code de la classe `Dessin` pour créez et afficher 100 instances de `Cercle`.
3. On ajoute maintenant dans la classe `Cercle` plusieurs méthodes qui nous seront utiles pour implémenter l'algorithme ci-dessus. Implémentez les méthodes suivantes :
   a. `public boolean intersecte(Cercle c)` renvoyant vrai si `this` et `c` s'intersectent (i.e. la distance entre les deux centres est inférieure à la somme des rayons)
   b. `private boolean estDansLeCadre(double largeur, double hauteur)` permettant de déterminer si le cercle est intégralement dans le cadre (la fenêtre de taille hauteur x largeur) ou non.
   c. `private boolean estValide(List<Cercle> liste, double largeur, double hauteur)` renvoyant vrai si le cercle actuel est *compatible* avec la liste de cercles (i.e. n'intersecte aucun Cercle de la liste) et est dans le cadre défini par largeur et hauteur.
   d. `private void placerAuHasard(List<Cercle> liste, double largeur, double hauteur)` qui positionne `this` au hasard dans une zone disponible.

4. Modifiez le constructeur de `Cercle` pour que le cercle ait un rayon de `0.1` et soit placé au hasard dans un endroit convenable.

Note

Vous pouvez commencer à tester vos premiers algorithmes maintenant (attention, fort risque de boucle infinie !). Pour "voir" les ronds sur le dessin, je vous conseille de faire en sorte que le rayon par défaut soit de 10 (à la place de 0.1) ET de n'ajouter que 5 ronds dans le cadre. Vous devez alors obtenir un dessin avec 5 ronds colorés, de même rayon, qui ne s'intersectent pas.

5. Définissez une méthode `public void grossir(List<Cercle> liste, double largeur, double hauteur)` qui augmente le rayon du cercle par pas de 0.1 jusqu'à ce qu'il ne soit plus possible de l'agrandir : soit il intersecterait un autre cercle de la liste, soit il dépasserait du cadre.
6. Modifiez le constructeur de `Cercle` pour que le cercle soit placé au hasard dans un endroit convenable et ait la plus grande taille possible.

Note

Vérifiez que vous avez bien remis la valeur du rayon par défaut à 0.1 et que votre code permet d'obtenir le dessin attendu soit 100 ronds colorés qui ne s'intersectent pas et qui sont *les plus grands possible*.

Amélioration du code

Bravo ! Votre dessin ressemble maintenant à ce qui est attendu. Mais l'algorithme de la méthode `grossir` n'est pas très satisfaisant : pour obtenir un *Cercle* de rayon 100 pixels, il faut faire 1000 itérations, qui chacune teste l'intersection avec tous les autres cercles ! On peut faire beaucoup mieux en utilisant un algorithme de dichotomie : on commence par essayer de faire grossir le rayon du cercle avec une **precision** de *min(largeur,hauteur)* , puis de la moitié, puis de la moitié, etc.

Voici l'algorithme:

```
rayon = 0.1
rayonMax = min(largeur,hauteur)
Pour precision de rayonMax à 0.1 en divisant par deux à chaque étape :
    Augmenter le rayon de precision si c'est possible
```

Implémentez cet algorithme.

Algorithmes

On souhaite avoir des informations sur les dessins générés : quelle est la taille du plus gros cercle, du plus petit, la taille moyenne, la surface totale des cercles, ...

Écrivez une bibliothèque `BibCercles` contenant les méthodes suivantes :

1. `public static Cercle getMin(List<Cercle> liste)` renvoyant le plus petit cercle de la liste,
2. `public static double surfaceTotale(List<Cercle> liste)` renvoyant la surface totale de tous les cercles,
3. Ajoutez d'autres méthodes (taille du cercle le plus gros, rayon moyen ...) et complétez l'affichage

Avec des rectangles maintenant

Et maintenant, essayez de faire la même chose que précédemment, pour générer un dessin composé de rectangles de tailles et couleurs aléatoires.