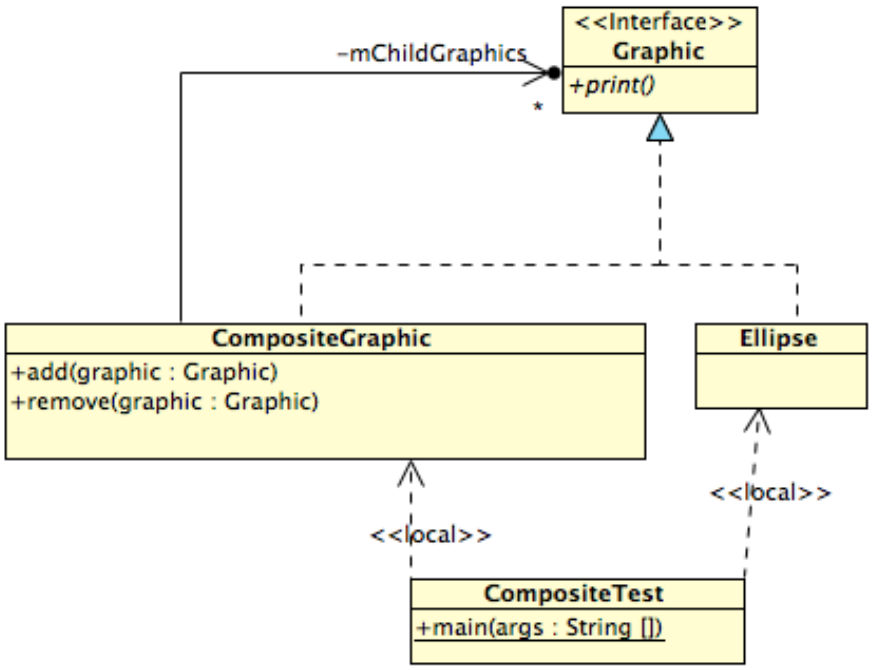
TD 6 - Patrons de conception

Exercice 1 : Graphic

- Expliquer le pattern utilisé dans l'exemple ci-dessous
- Le code Java de la classe de test est donné en partie.
- Donner le code Java de l'application sachant que la méthode print() est réduite à imprimer le type de l'objet concerné.
- Compléter le code de la classe de test.
- Que donne l'exécution de la classe de test.




```
package composite;
public class CompositeTest{

    public static void main(String[] args) {
        //create four ellipses


        //create three composite graphics


        //Composes the graphics
        graphic1.add(ellipse1);
        graphic1.add(ellipse2);
        graphic1.add(ellipse3);

        graphic2.add(ellipse4);

        graphic.add(graphic1);
        graphic.add(graphic2);

        graphic.print();
    } }
```

Exercice 2 : Strategy

On considère un client qui peut payer sa note de restaurant de différentes façons : carte bleu, en espèce ou par chèque.

Le client doit pouvoir choisir une des trois façons de payer.
Pour l'instant le programme Java correspondant est le suivant :

```
public class CustomerOld {
  /**
   * paymentStrategy : 1 Cash, 2 Check, 3 DebitCard
  */
    private int paymentStrategy;

    public CustomerOld(int paymentStrategy) {
           this.paymentStrategy = paymentStrategy;
    }

     public void pay(int amount) {
       if (paymentStrategy == 1)
              this.payCash(amount);
       else if ( paymentStrategy == 2)
             this.payCheck(amount);
       else if (paymentStrategy == 3)
            this.payDebitCard(amount);
       else System.out.println ("Wrong number "+ this.paymentStrategy);
   }

    public void payCash(int amount) {
       System.out.println("Charged "+amount+" using Cash...");
       System.out.println("Thank you!!");
    }

      public void payDebitCard(int amount) {
          System.out.println("Charging "+amount+" using Debit Card...");
          System.out.println("Thank you!!");
   }

      public void payCheck(int amount) {
         System.out.println("Charging "+amount+" using Check...");
        System.out.println("Thank you!!");
    }
```

```
public class TestClassOld {
    public static void main(String[] args) {
          CustomerOld customer1 = new CustomerOld(3);
          customer1.pay(1000);

          CustomerOld customer2 = new CustomerOld(1);
          customer2.pay(5000);
   }
}
```

Cette solution n'est pas très élégante et fait appel à des valeurs numériques. Le patron Strategy semble un bon candidat pour répondre au problème puisque l'on a 3 stratégies pour régler la note du restaurant.

1 – Commencer par créer une interface de nom IPaymentStrategy qui spécifie la méthode de paiement.
2- Donner le diagramme de classes du pattern Srategy appliqué à cet exemple..
3- Donner le code Java correspondant.
4- Modifier la classe TestClassOld pour qu'elle fonctionne avec les autres classes.

Exercice 3 : Observer (à finir en TP)

Nous souhaitons mettre à jour automatiquement l'affichage de conditions météorologiques.
Pour cela nous avons

- une classe **`WeatherData`** qui est utilisée pour stocker la température, l'humidité et la pression à un moment donné.
- Une classe **`CurrentConditionsDisplay`** dont le rôle est d'afficher les données de WeatherData.
- Une classe **`WeatherStation`** qui crée les objets WeatherData et CurrentConditionsDisplay et qui lance trois séries de mesures.

Il s'agit d'appliquer le patron Observer à cet exemple :

- Commencer par dessiner le diagramme de classes qui pourra être modifié ensuite avec Observer
- Modifier le code pour implémenter le patron Observer afin que l'affichage se fasse bien automatiquement à chaque changement de mesures.

```
package weather;

public class WeatherStation {
   public static void main(String[] args) {
       WeatherData weatherData = new WeatherData();
       CurrentConditionsDisplay currentDisplay = new
                               CurrentConditionsDisplay(weatherData);
       weatherData.setMeasurements(12, 65, 1010);
       currentDisplay.weatherChanged();
       weatherData.setMeasurements(25, 40, 1030);
       currentDisplay.weatherChanged();
       weatherData.setMeasurements(3, 90, 980);
       currentDisplay.weatherChanged();
   }
}
```

```
package weather;

public class WeatherData {
       private float temperature;
       private float humidity;
       private float pressure;

       public WeatherData() {
       }


       public void setMeasurements(float temperature, float humidity,
                                                       float pressure) {
         this.temperature = temperature;
         this.humidity = humidity;
         this.pressure = pressure;

       }

       public float getTemperature() {
            return temperature;
      }
```

```
       public float getHumidity() {
          return humidity;
       }

       public float getPressure() {
            return pressure;
      }
}
```

```
package weather;

public class CurrentConditionsDisplay {
      private float temp = 0;
      private float humidity;
      private float pressure;

      private WeatherData weatherData;

      public CurrentConditionsDisplay(WeatherData weatherData) {
            this.weatherData = weatherData;
      }

      public void weatherChanged() {
         temp =weatherData.getTemperature();
         humidity = weatherData.getHumidity();
         pressure= weatherData.getPressure();
         this.display();
      }

      public void display() {
            System.out.println("Current Conditions = " +
             " + temperature :" + temp
               + "/" + "humidity : "+ humidity
               + "/" + "pressure : "+ pressure);
      }

}
```