

Ampoules

Tâche 1 : Créer une classe Lightbulb.

```
public class LightBulb {  
}
```

Tâche 2 : Définissez les signatures (et uniquement les signatures pas le code) des méthodes pour cette classe :

- `getPower()`, `getColor()` et `isOn()` pour accéder aux différents attributs d'une ampoule;
- `on()` et `off()` pour allumer ou éteindre une ampoule;
- `public String toString()` qui fournit une chaîne de caractères décrivant l'ampoule.

Pour que votre code compile, mettez pour le moment des instructions `return` par défaut (0, `false` ou `null` suivant que le retour de la méthode soit un nombre, un booléen ou un objet).

```
public class LightBulb {  
    int getPower(){  
        return 0;  
    }  
    String getColor(){  
        return null;  
    }  
    boolean isOn(){  
        return false;  
    }  
    void on(){  
  
    }  
    void off(){  
  
    }  
    public String toString(){  
        return null;  
    }  
}
```

Tâche 3 : Définissez dans la classe les attributs nécessaires pour ces méthodes ainsi que le constructeur.

Quel est selon vous le type le plus pertinent pour représenter l'état allumé ou éteint d'une ampoule ?

Pour représenter l'état allumé ou éteint d'une ampoule, le type le plus adapté est `boolean` avec valeur `true` pour une ampoule allumée et `false` pour une ampoule éteinte.

```
public class LightBulb {
    boolean isOn;
    int power;
    String color;

    public LightBulb(boolean isOn, int power, String color) {
        this.isOn = isOn;
        this.power = power;
        this.color = color;
    }

    int getPower(){
        return 0;
    }
    String getColor(){
        return null;
    }
    boolean isOn(){
        return false;
    }
    void on(){

    }
    void off(){

    }
    public String toString(){
        return null;
    }
}
```

Tâche 4 : Codez le corps des méthodes de la classe et enlever les retours par défaut.

```
public class LightBulb {
    boolean isOn;
    int power;
    String color;

    public LightBulb(boolean isOn, int power, String color) {
        this.isOn = isOn;
        this.power = power;
        this.color = color;
    }
}
```

```

int getPower(){
    return power;
}
String getColor(){
    return color;
}
boolean isOn(){
    return isOn;
}
void on(){
    isOn = true;
}
void off(){
    isOn = false;
}
public String toString(){
    String state;
    if(isOn())
        state = "allumée";
    else
        state = "éteinte";
    return "Ampoule " + getColor() + " (" + getPower() + "W) : " + state;
}
}

```

Tâche 5 : Construisez des instances de cette classe et manipulez les pour faire des tests sur le fonctionnement des méthodes que vous avez codé.

```

public class AppLightBulb {
    public static void main(String[] args){
        LightBulb l1, l2;
        l1 = new LightBulb(true, 100, "bleue");
        l2 = new LightBulb(false, 200, "rouge");
        System.out.println(l1.getColor());
        System.out.println(l2.isOn());
        System.out.println(l1.getPower());
        System.out.println(l1);
        System.out.println(l2);
        l1.off();
        l2.on();
        System.out.println(l1);
        System.out.println(l2);
    }
}

```

Interrupteurs

On s'intéressera ici à la modélisation d'interrupteurs (*switch*) électriques qui permettent l'allumage et l'extinction d'une (et une seule) ampoule donnée. L'ampoule contrôlée par l'interrupteur est un attribut de cet objet. Sa valeur est fournie en paramètre lors de la construction de l'objet interrupteur. On doit pouvoir accéder à l'ampoule contrôlée par l'interrupteur. On peut appuyer sur un interrupteur ce qui a pour effet, à chaque appui, de faire passer l'ampoule de éteinte à allumée ou inversement.

Tâche 6 : Créez une classe `Switch`.

Tâche 7 : Définissez les signatures (et uniquement les signatures pas le code) des méthodes pour cette classe : `getLightbulb()` et `push()`

Tâche 8 : Définissez les attributs (l'état) des instances de cette classe et son constructeur.

Tâche 9 : Codez le corps des méthodes de la classe et enlever les retours par défaut.

Tâche 10 : Construisez des instances de cette classe et manipulez les pour faire des tests sur le fonctionnement des méthodes que vous avez codé.

Guirlandes

Une guirlande électrique (*light string*) est composée d'un certain nombre d'ampoules électriques. Il est possible d'allumer et d'éteindre la guirlande, c'est-à-dire que l'on allume ou éteint toutes les ampoules de la guirlande «en même temps». On souhaite créer une classe `LightString` pour modéliser de telles guirlandes. Cette classe doit respecter les contraintes décrites ci-dessous. Vous pouvez ajouter à la classe d'autres méthodes qui vous semblent pertinentes.

Les contraintes de la classe `LightString` sont :

- les ampoules qui composent la guirlande sont des objets du type `Lightbulb`.
- le nombre d'ampoules de la guirlande est défini à sa création et les ampoules qui composent la guirlande sont créées au moment la création de la guirlande.

Initialement toutes ces ampoules sont éteintes, de couleur blanche et ont une puissance de 1W. Vous utiliserez un attribut de type tableau d'objets `LightBulb` pour gérer ces ampoules au sein de la guirlande.

- on veut disposer d'une méthode qui permette d'obtenir la *i*-ème ampoule de la guirlande, le résultat de la méthode est null si l'indice *i* fourni en paramètre n'est pas valide.
 1. Écrivez la signature de la méthode.
 2. Écrivez sa documentation.
 3. Codez cette méthode.
- on veut pouvoir changer une des ampoules de la guirlande grâce à la méthode suivante `void changeLightbulb(int i, Lightbulb theBulb)` qui change la *i*-ème ampoule de la guirlande.

Tâche 11 : Donnez le code de cette méthode.

Tâche 12 : Donnez le code des méthodes `on()` et `off()` qui permettent de d'allumer ou d'éteindre toutes les ampoules de la guirlande.

Tâche 13 : Donnez le code de la méthode `getConsumedPower()` fournit comme résultat la puissance totale exprimée en Watts consommée par les ampoules de la guirlande lorsqu'elle est allumée, et 0 sinon.