pi-8. クラス設計

トピックス：クラス設計，オブジェクトの状態と状態変化，メソッド内でのみ使用する変数，抽象化の組み合わせ

URL: https://www.kkaneko.jp/cc/pi/index.html

（Java の基本）

金子邦彦

アウトライン

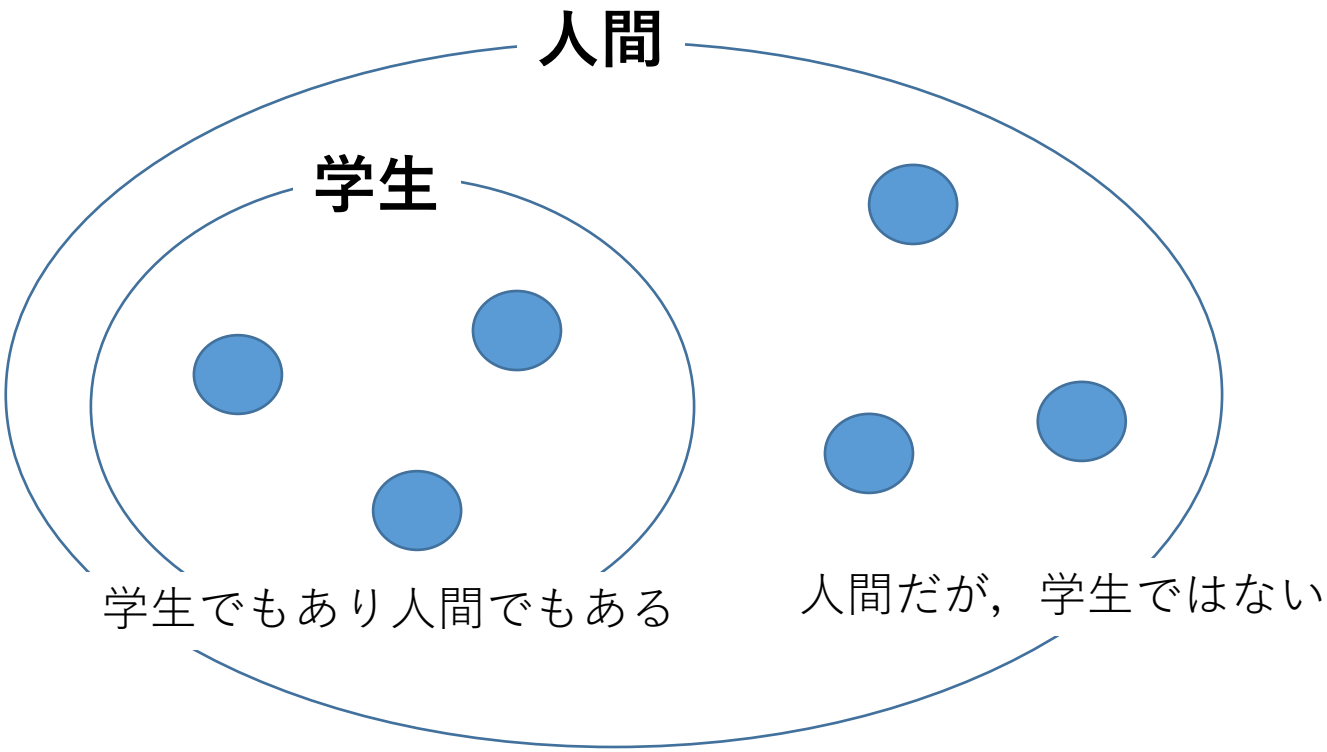
番号	項目
	復習
8-1	クラスの設計の例
8-2	オブジェクトの状態と状態変化
8-3	演習
8-4	メソッド内でのみ使用する変数
8-5	抽象化の組み合わせ

各自、資料を読み返したり、課題に取り組んだりも行う

この授業では、**Java** を用いて基礎を学び、マスターする

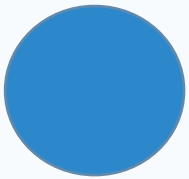
クラスとオブジェクト

クラスは，**同じ種類のオブジェクトの集まり**と考えることができる



Java のクラス
クラス Ball

オブジェクト



半径 1，場所（8, 10）
色 blue

オブジェクト

半径 3，場所（2, 4）
色 green

```java
class Ball {
    double x;
    double y;
    double r;
    String color;
    public Ball(double x, double y, double r, String color) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.color = color;
    }
    public void printout() {
        System.out.println(this.x);
        System.out.println(this.y);
        System.out.println(this.r);
        System.out.println(this.color);
    }
}
```

クラス定義のプログラム

```java
Ball a = new Ball(8, 10, 1, "blue");
Ball b = new Ball(2, 4, 3, "green");
```

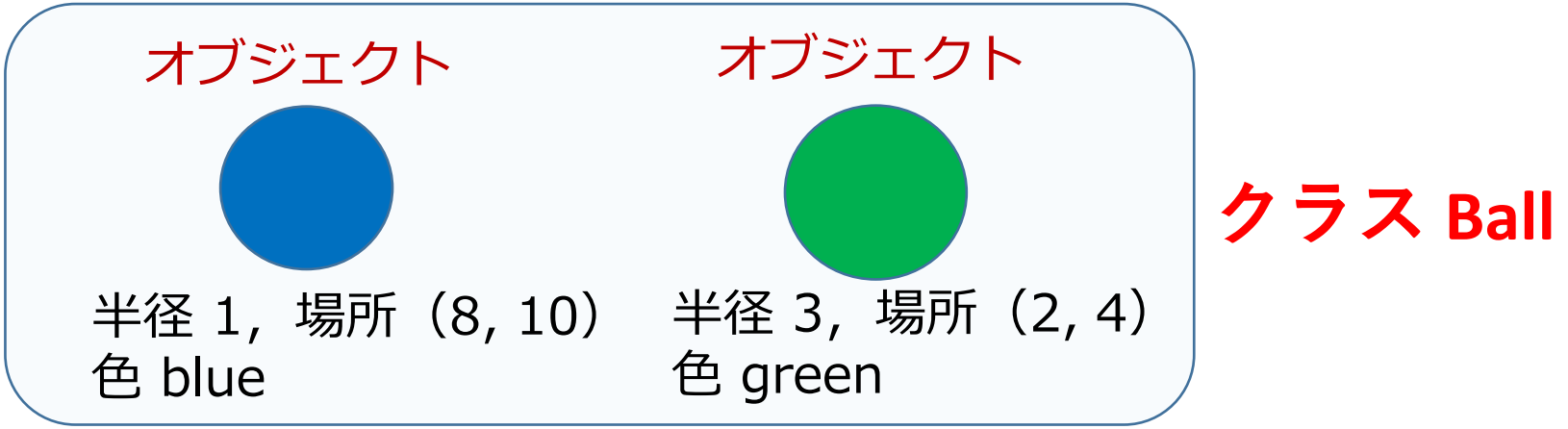
オブジェクト生成のプログラム

```java
a.printout();
b.printout();
```

メソッドアクセスのプログラム

クラス，属性やメソッドへのアクセス，this

- **クラス**は，**同じ種類のオブジェクトの集まり**と考えることができる



- **メソッド定義内**で，その**メソッドが所属するクラス**で定義された**属性**や**メソッド**にアクセスするときは **this ＋「.」**

```
public Ball(double x, double y, double r, String color) {
    this.x = x;
    this.y = y;
    this.r = r;
    this.color = color;
}
```

Java Tutor の起動

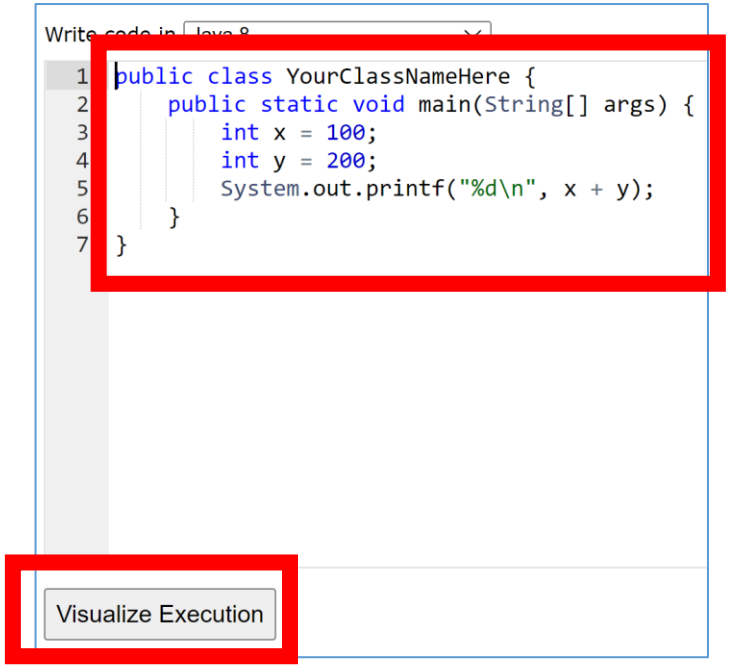
① **ウェブブラウザ**を起動する

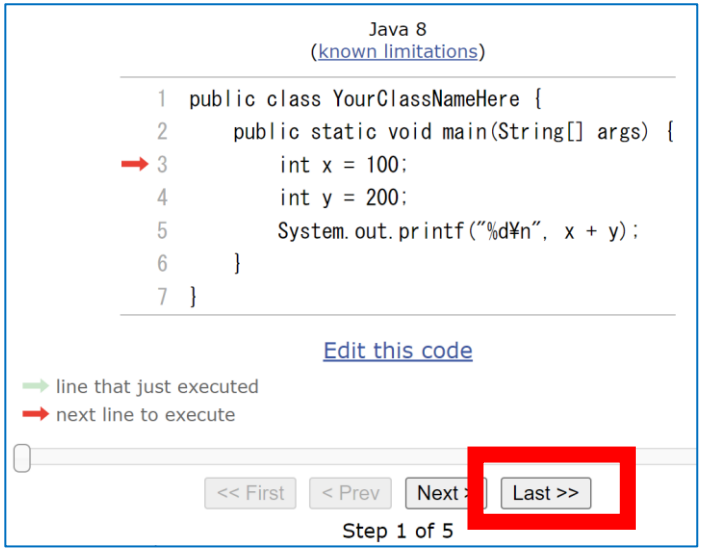
② **Java Tutor** を使いたいので，次の URL を開く

http://www.Javatutor.com/

③ 「**Java**」をクリック　⇒ **編集画面**が開く

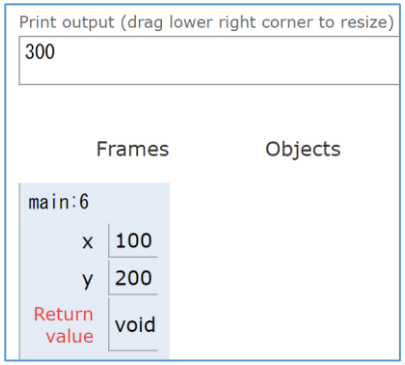
Java Tutor でのプログラム実行手順

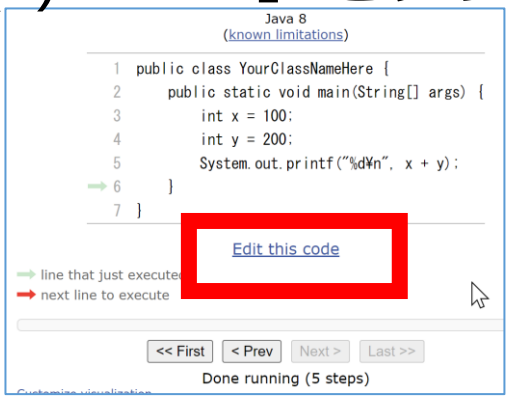




(1)「**Visualize Execution**」をクリックして**実行画面**に切り替える

(2)「**Last**」をクリック.





(3) **<u>実行結果を確認</u>**する.

(4)「**Edit this code**」をクリックして**編集画面**に戻る

Java Tutor 使用上の注意点①

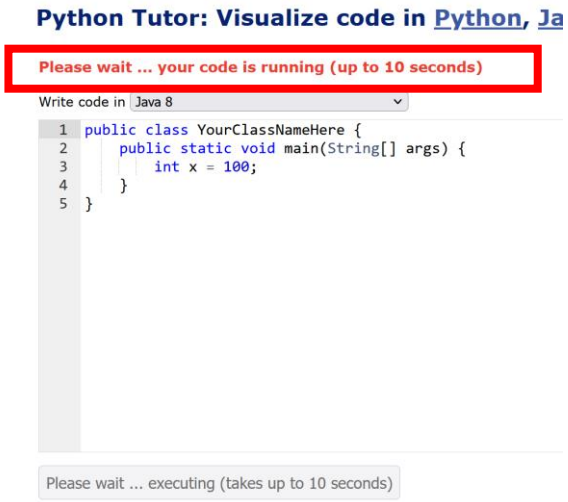
- **実行画面で，次のような赤の表示が出ることがある → 無視してよい**

  過去の文法ミスに関する確認表示

  邪魔なときは「**Close**」

Java Tutor 使用上の注意点②

「**please wait ... executing**」のとき，**１０秒ほど待つ**.



→　**混雑しているとき**は，**「Server Busy・・・」というメッセージが出る**ことがある.

混雑している．**少し（数秒から数十秒）待つ**と自動で表示が変わる（変わらない場合には，操作をもう一度行ってみる）

8-1. クラス設計の例

クラスの設計例 ①

1. **クラス名は Signal**
2. **属性 color は文字列である.**
3. **属性 color は “red”, “yellow”, “blue” の 3通り.**

```
class Signal {
    String color;
    public Signal(String color) {
        this.color = color;
    }
}
public class YourClassNameHere {
    public static void main(String [] args) {
        Signal s = new Signal("red");
    }
}
```

クラス定義

クラスの設計例 ①

1. **クラス名は Signal**
2. **属性 color は文字列である.**
3. **属性 color は “red”, “yellow”, “blue” の 3通り.**

```
class Signal {
    String color;
    public Signal(String color) {
        this.color = color;
    }
}
public class YourClassNameHere {
    public static void main(String [] args) {
        Signal s = new Signal("rrd");
    }
}
```

クラス定義

書き間違ってしまっても、実行できてしまうので、もろいプログラム

書き間違い

あとで書き間違いを探すのは面倒そう

演習

資料：**13 ～ 15**

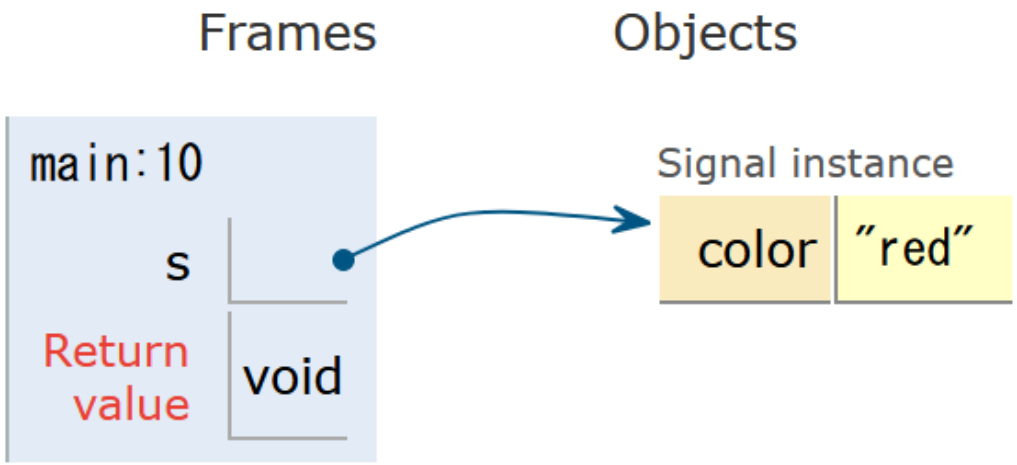
【トピックス】

- **クラス定義**
- **class**
- **オブジェクト生成（コンストラクタ）**

① Java Tutor のエディタで次のプログラムを入れる

```
1  class Signal {
2      String color;
3      public Signal(String color) {
4          this.color = color;
5      }
6  }
7  public class YourClassNameHere {
8      public static void main(String [] args) {
9          Signal s = new Signal("red");
10     }
11 }
```

② 実行し，結果を確認する．

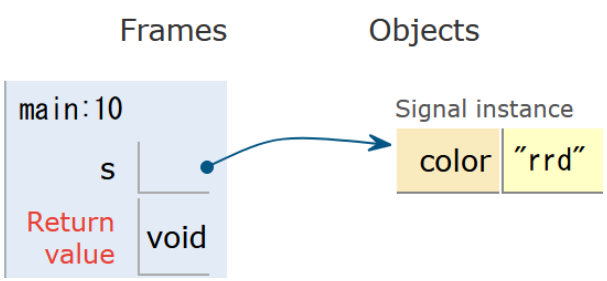


「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

③ 「red」を「rrd」と書き換えても実行できる

（あとで使うので消さないこと）

```
1  class Signal {
2      String color;
3      public Signal(String color) {
4          this.color = color;
5      }
6  }
7  public class YourClassNameHere {
8      public static void main(String [] args) {
9          Signal s = new Signal("rrd");
10     }
11 }
```



「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

クラスの設計例 ②

1. **クラス名は Signal**
2. **属性 color は文字列である.**
3. **属性 color は “red”, “yellow”, “blue” の 3通り.**

```
class Signal {
    String color;
    public Signal() {};
    public void red() { this.color = "red"; };
    public void yellow() { this.color = "yellow"; };
    public void blue() { this.color = "blue"; };
}
public class YourClassNameHere {
    public static void main(String [] args) {
        Signal s = new Signal();
        s.red();
    }
}
```

新しいクラス定義

- コンストラクタでは、何も行わない
- メソッド red, yellow, blue を新設
- 「void」は、「メソッドの返り値なし」の意味

演習

資料：**18 ～ 19**

【トピックス】

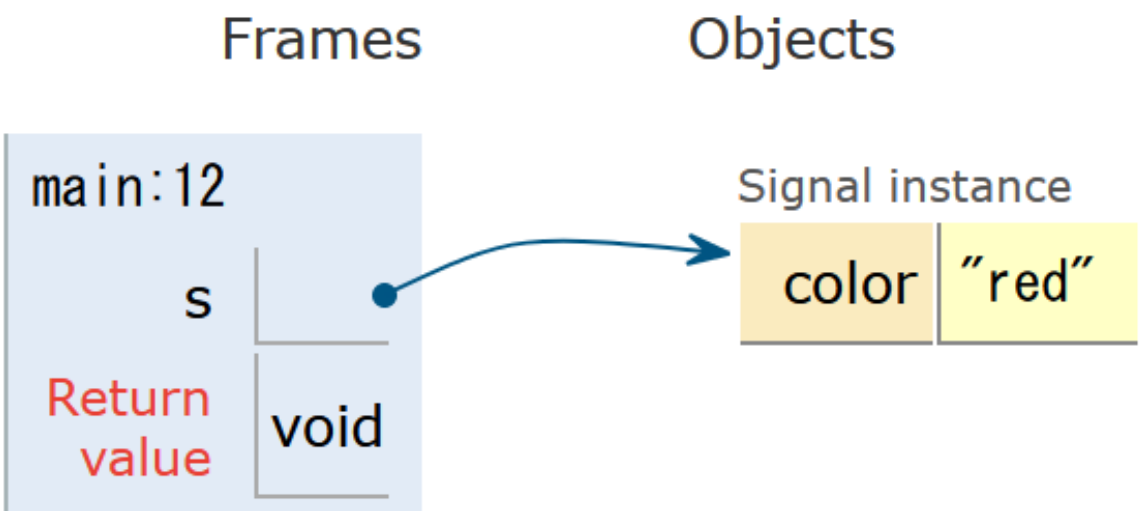
・クラス設計

① Java Tutor のエディタで次のプログラムを入れる

```
1   class Signal {
2       String color;
3       public Signal() {};
4       public void red() { this.color = "red"; };
5       public void yellow() { this.color = "yellow"; };
6       public void blue() { this.color = "blue"; };
7   }
8   public class YourClassNameHere {
9       public static void main(String [] args) {
10          Signal s = new Signal();
11          s.red();
12      }
13  }
```

② 実行し，結果を確認する

（あとで使うので消さないこと）



「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

まとめ
２つの方法の比較

```
class Signal {
    String color;
    public Signal(String color) {
        this.color = color;
    }
}
```

クラス①

```
class Signal {
    String color;
    public Signal() {};
    public void red() { this.color = "red"; };
    public void yellow() { this.color = "yellow"; };
    public void blue() { this.color = "blue"; };
}
```

クラス②

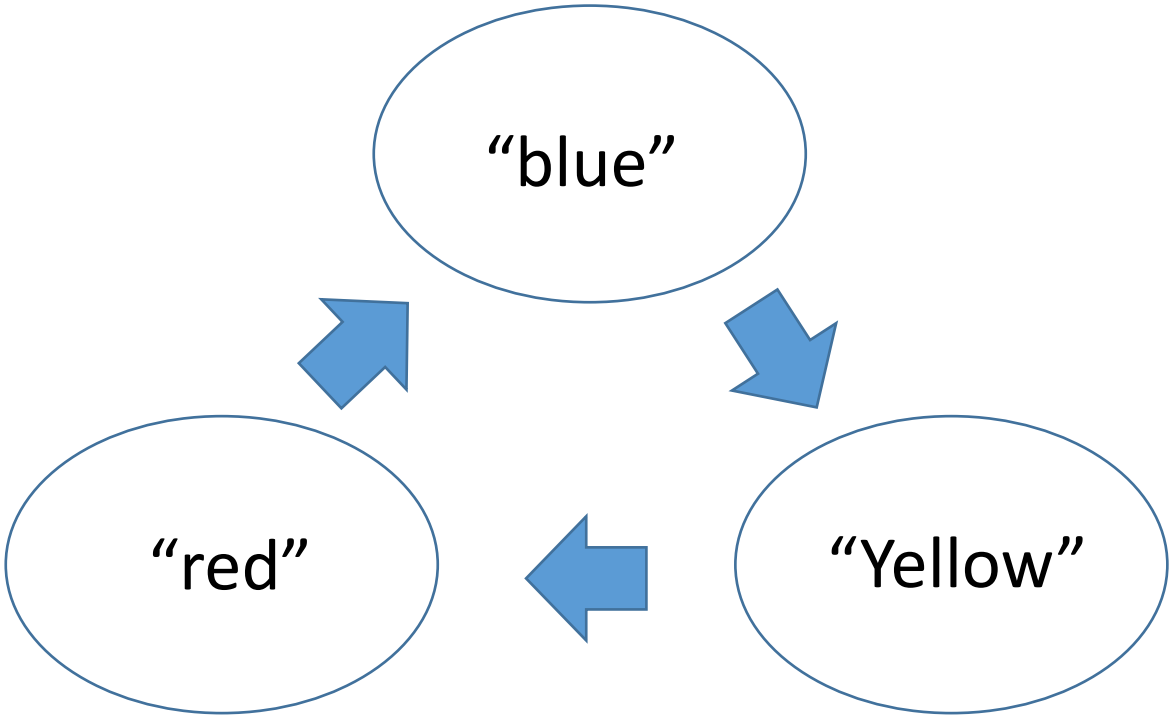
プログラムの長さ：　①短い、②長い
不正なデータの混入可能性：　①あり、②なし

8-2. オブジェクトの状態と状態変化

オブジェクトの状態と状態変化

信号の**状態**を、**属性 color** で扱う

- **属性 color** は “red”, “yellow”, “blue” の 3通り.
- **属性 color** は、次のように**変化**する



状態変化のメソッドの例

```
class Signal {
    String color;
    public Signal() {};
    public void red() { this.color = "red"; };
    public void yellow() { this.color = "yellow"; };
    public void blue() { this.color = "blue"; };
    public void go() {
        if (this.color.equals("blue")) { this.yellow(); }
        else if (this.color.equals("yellow")) { this.red(); }
        else if (this.color.equals("red")) { this.blue(); }
    }
}
```

※ equals メソッドは文字列の比較

演習

資料：**25 ～ 26**

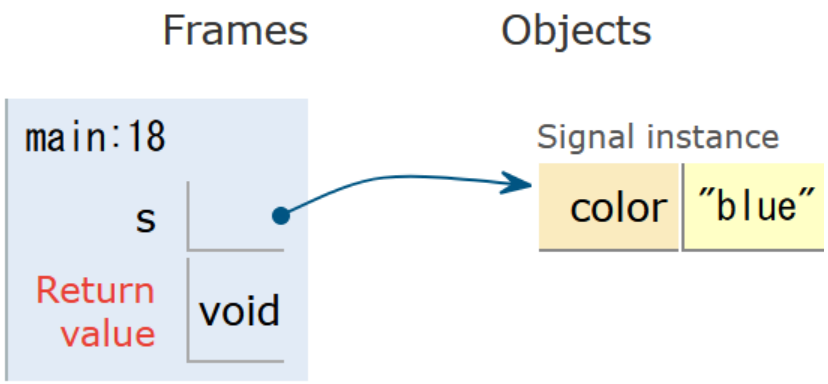
【トピックス】

・オブジェクトの状態変化

① Java Tutor のエディタで次のプログラムを入れる

```
1  class Signal {
2      String color;
3      public Signal() {};
4      public void red() { this.color = "red"; };
5      public void yellow() { this.color = "yellow"; };
6      public void blue() { this.color = "blue"; };
7      public void go() {
8          if (this.color.equals("blue")) { this.yellow(); }
9          else if (this.color.equals("yellow")) { this.red(); }
10         else if (this.color.equals("red")) { this.blue(); }
11     }
12 }
13 public class YourClassNameHere {
14     public static void main(String [] args) {
15         Signal s = new Signal();
16         s.red();
17         s.go();
18     }
19 }
```

② 実行し，結果を確認する



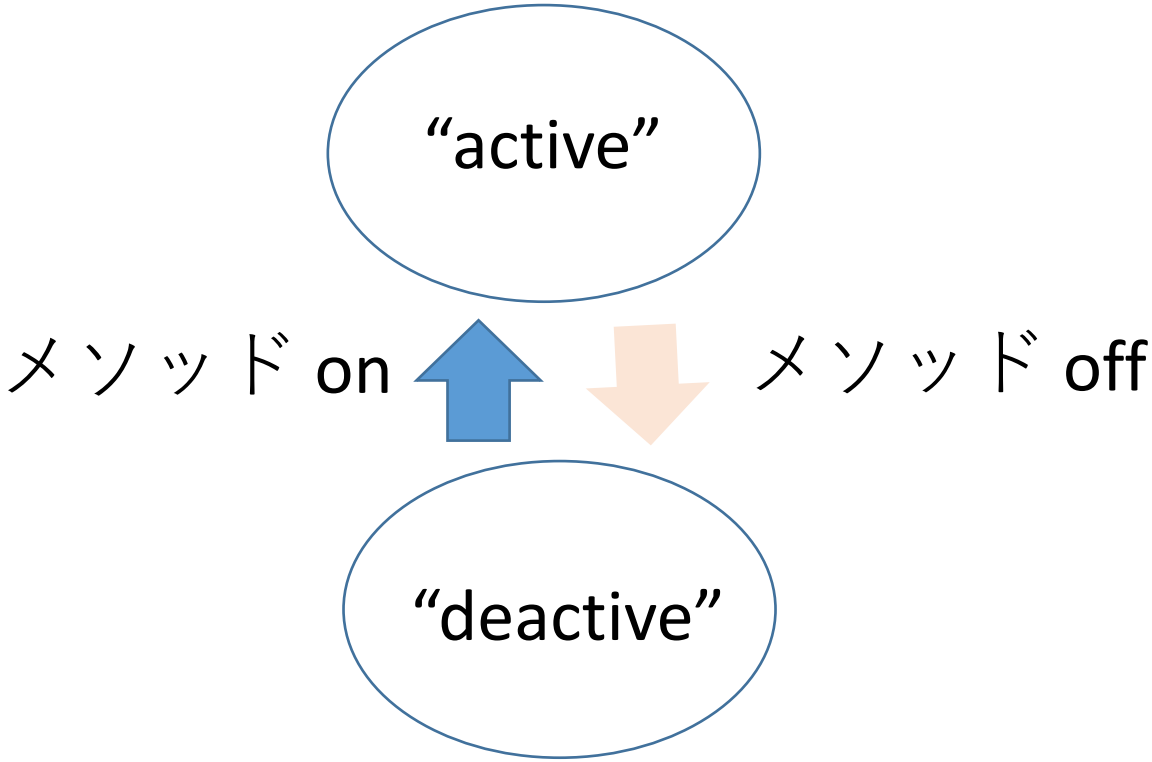
「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

8-3. 演習

オブジェクトの状態と状態変化

製品の**状態**を、**属性 s** で扱う

- **属性 s** は “active”, “deactive” の 2通り.
- **属性 s** は、次のように**変化**する



ソースコード

```
class Product {
    String s;
    public Product() {};
    public void active() { this.s = "active"; };
    public void deactive() { this.s = "deactive"; };
    public void on() {
        if (this.s.equals("deactive")) { this.active(); }
    }
    public void off() {
        if (this.s.equals("active")) { this.deactive(); }
    }
}
public class YourClassNameHere {
    public static void main(String [] args) {
        Product p = new Product();
        p.deactive();
        p.on();
    }
}
```

演習

資料：**31 ～ 32**

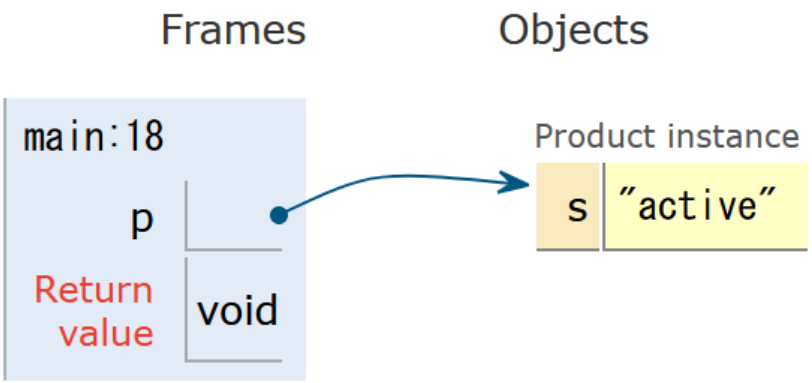
【トピックス】

- **オブジェクトの状態と状態変化**

① Java Tutor のエディタで次のプログラムを入れる

```java
class Product {
    String s;
    public Product() {};
    public void active() { this.s = "active"; };
    public void deactive() { this.s = "deactive"; };
    public void on() {
        if (this.s.equals("deactive")) { this.active(); }
    }
    public void off() {
        if (this.s.equals("active")) { this.deactive(); }
    }
}
public class YourClassNameHere {
    public static void main(String [] args) {
        Product p = new Product();
        p.deactive();
        p.on();
    }
}
```

② 実行し，結果を確認する



「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

8-4. メソッド内でのみ使用する変数

- メソッド **foo** : 変数 **a** を使用
- メソッド **main**: 変数 **p** を使用

```
1  public class YourClassNameHere {
2      public static double foo(double a) {
3          return a * 1.1;
4      }
5      public static void main(String[] args) {
6          double p;
7          p = 120;
8          System.out.printf("%f\n", foo(p));
9          p = 200;
10         System.out.printf("%f\n", foo(p));
11     }
12 }
```

演習

資料：**36 ～ 37**

【トピックス】

- **メソッド内で使用される変数**は，**メソッドの実行終了**とともに，**自動で消える**

Java Tutor のエディタで次のプログラムを入れ，実行し，結果を確認する（あとで使うので消さないこと）

```
1  public class YourClassNameHere {
2      public static double foo(double a) {
3        return a * 1.1;
4      }
5      public static void main(String[] args) {
6        double p;
7        p = 120;
8        System.out.printf("%f\n", foo(p));
9        p = 200;
10       System.out.printf("%f\n", foo(p));
11     }
12 }
```

Print output (drag lower right corne

```
132.000000
220.000000
```

「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

- **変数 p は、main の中でのみ利用できる**．foo の中では利用**できない**から、次のプログラムは動かない

```java
public class YourClassNameHere {
    public static double foo(double a) {
        System.out.printf("%f\n", p);
        return a * 1.1;

    }
    public static void main(String [] args) {
        double p;
        p = 120;
        System.out.printf("%f\n", foo(p));
        p = 200;
        System.out.printf("%f\n", foo(p));
    }
}
```

Error: cannot find symbol
  symbol: variable p
  location: class YourClassNameHere

8-5. 抽象化の組み合わせ

式の抽象化

5 * 100 * 1.1

12 * 100 * 1.1

類似した複数の**式**

a * 100 * 1.1

変数を使って，複数の**式**を１つにまとめる（**抽象化**）

抽象化の組み合わせ

5 * **100** * 1.1

12 * **100** * 1.1

8 * **200** * 1.1

16 * **200** * 1.1

→

b * **100** * 1.1

c * **200** * 1.1

抽象化

→

a * 1.1

抽象化

類似した複数の**式**

5 * 100 * 1.1

12 * 100 * 1.1

8 * 200 * 1.1

16 * 200 * 1.1

b * 100 * 1.1

c * 200 * 1.1

抽象化

a * 1.1

抽象化

```
public static double y(double b) {
    return x(b * 100);
}
public static double z(double c) {
    return x(c * 200);
}
```

```
public static double x(double a) {
  return a * 1.1;
}
```

演習

資料：**43 ～ 44**

【トピックス】

- **抽象化**

① Java Tutor のエディタで次のプログラムを入れる

```java
public class YourClassNameHere {
    public static double x(double a) {
      return a * 1.1;
    }
    public static double y(double b) {
        return x(b * 100);
    }
    public static double z(double c) {
        return x(c * 200);
    }
    public static void main(String [] args) {
        System.out.printf("%f\n", y(5));
        System.out.printf("%f\n", y(12));
        System.out.printf("%f\n", z(8));
        System.out.printf("%f\n", z(16));
    }
}
```

② 実行し，結果を確認する

Print output (drag

```
550.000000
1320.000000
1760.000000
3520.000000
```

「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

8-2. のプログラム

```
class Signal {
    String color;
    public Signal() {};
    public void red() { this.color = "red"; };
    public void yellow() { this.color = "yellow"; };
    public void blue() { this.color = "blue"; };
    public void go() {
        if (this.color.equals("blue")) { this.yellow(); }
        else if (this.color.equals("yellow")) { this.red(); }
        else if (this.color.equals("red")) { this.blue(); }
    }
}
public class YourClassNameHere {
    public static void main(String [] args) {
        Signal s = new Signal();
        s.red();
        s.go();
    }
}
```

8-3. のプログラム

```
class Product {
    String s;
    public Product() {};
    public void active() { this.s = "active"; };
    public void deactive() { this.s = "deactive"; };
    public void on() {
        if (this.s.equals("deactive")) { this.active(); }
    }
    public void off() {
        if (this.s.equals("active")) { this.deactive(); }
    }
}
public class YourClassNameHere {
    public static void main(String [] args) {
        Product p = new Product();
        p.deactive();
        p.on();
    }
}
```

8-4. のプログラム

```
public class YourClassNameHere {
    public static double foo(double a) {
        return a * 1.1;
    }
    public static void main(String [] args) {
        double p;
        p = 120;
        System.out.printf("%f¥n", foo(p));
        p = 200;
        System.out.printf("%f¥n", foo(p));
    }
}
```

8-5 のプログラム

```
public class YourClassNameHere {
    public static double x(double a) {
      return a * 1.1;
    }
    public static double y(double b) {
        return x(b * 100);
    }
    public static double z(double c) {
        return x(c * 200);
    }
    public static void main(String [] args) {
        System.out.printf("%f¥n", y(5));
        System.out.printf("%f¥n", y(12));
        System.out.printf("%f¥n", z(8));
        System.out.printf("%f¥n", z(16));
    }
}
```

全体まとめ

- **オブジェクト**の**属性**に、不正なデータが入らないように、**クラスの設計**、**メソッド**を工夫できる
- **オブジェクト**の**属性**と**メソッド**を、**状態**、**状態変化**のようにとらえることができる場合がある
- **抽象化**には**バリエーションがある。**（新しい種類の抽象化を説明します）

関連ページ

- **Java プログラミング入門**

GDB online を使用

https://www.kkaneko.jp/cc/ji/index.html

- **Java の基本**

Java Tutor, GDB online, VisuAlgo を使用

https://www.kkaneko.jp/cc/pi/index.html

- **Java プログラム例**

https://www.kkaneko.jp/pro/java/index.html