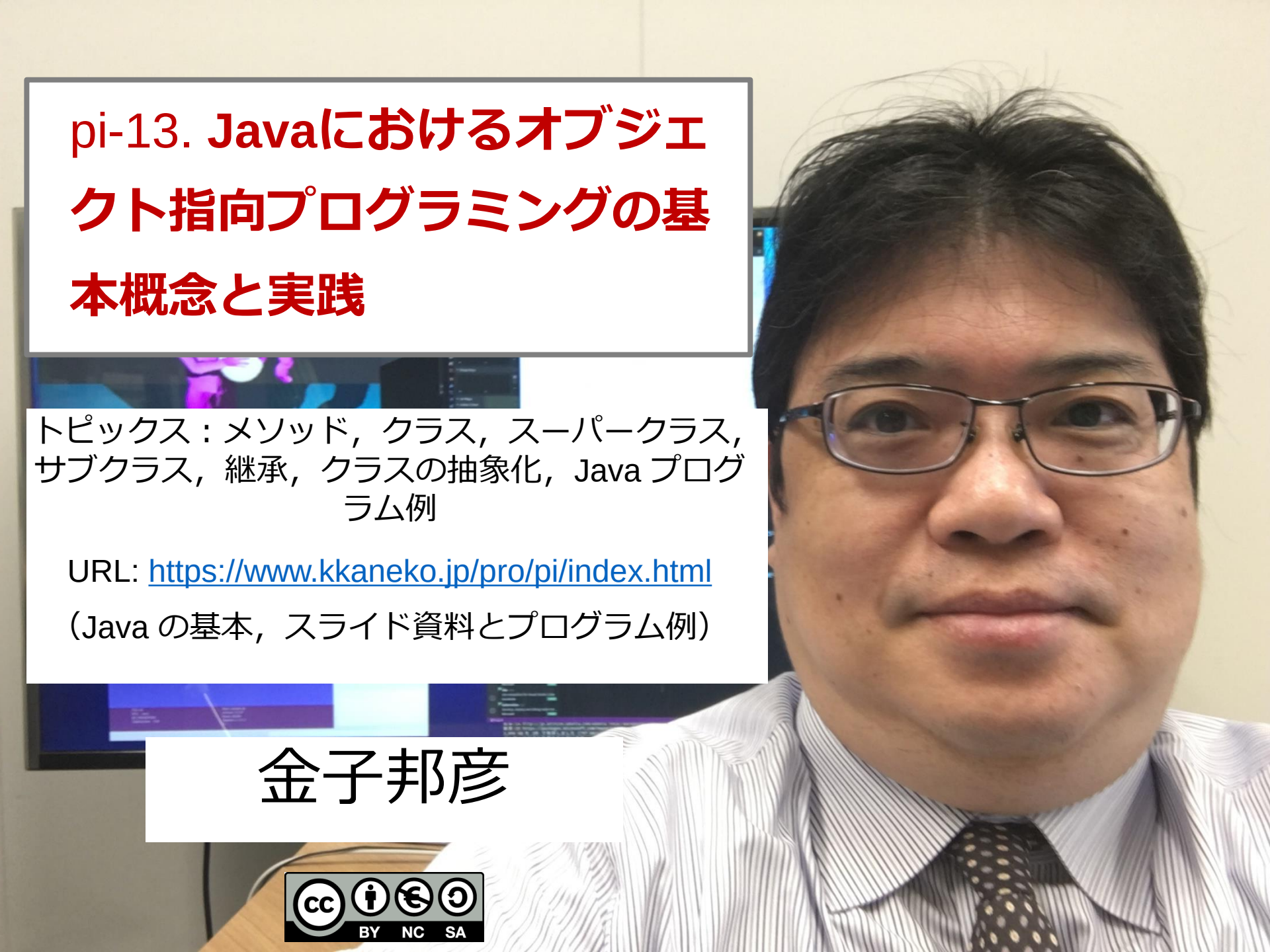
pi-13. Javaにおけるオブジェクト指向プログラミングの基本概念と実践

トピックス：メソッド，クラス，スーパークラス，サブクラス，継承，クラスの抽象化，Java プログラム例

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

（Java の基本，スライド資料とプログラム例）



金子邦彦

アウトライン

番号	項目
	復習
13-1	メソッド
13-2	クラス
13-3	スーパークラス、サブクラス、継承
13-4	クラスの抽象化
13-5	Java のプログラム例

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

Java Tutor の起動

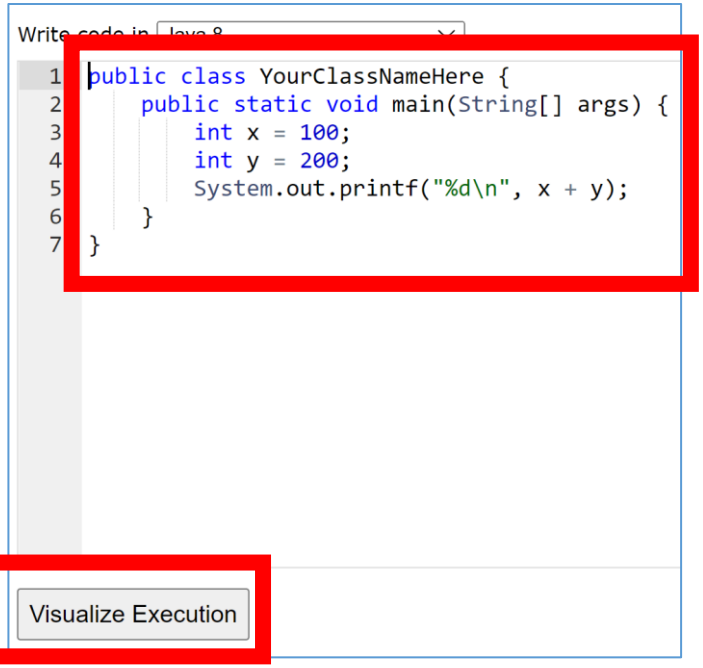
① **ウェブブラウザ**を起動する

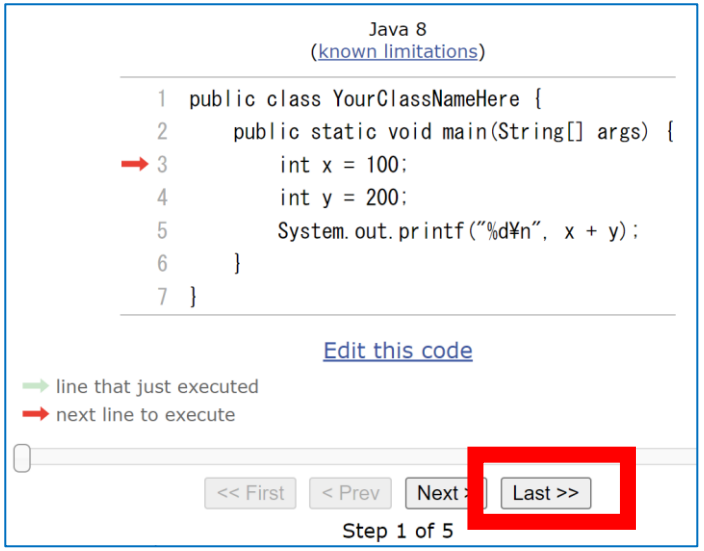
② **Java Tutor** を使いたいので，次の URL を開く

http://www.pythontutor.com/

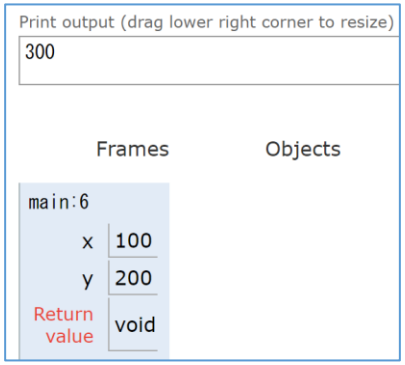
③ 「**Java**」をクリック　⇒ **編集画面**が開く

Java Tutor でのプログラム実行手順



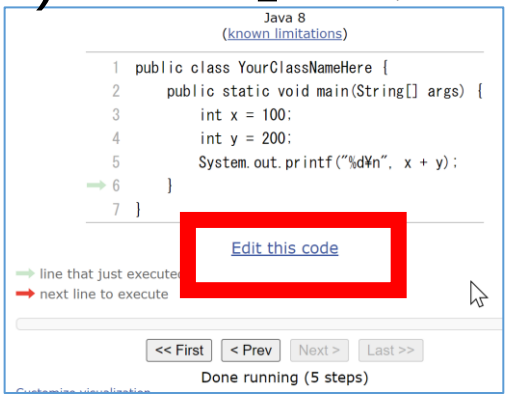


(1)「**Visualize Execution**」をクリックして**実行画面**に切り替える



(3) **<u>実行結果を確認</u>**する.

(2)「**Last**」をクリック.



(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

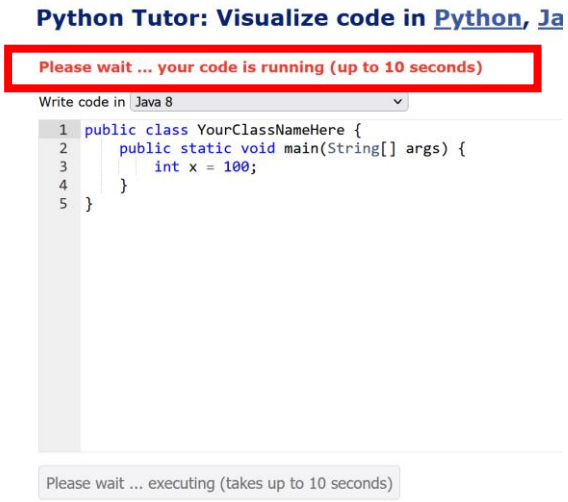
- **実行画面で，次のような赤の表示が出ることがある → 無視してよい**

  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）

13-1. メソッド

オブジェクトとメソッド

hero.moveDown()

hero　　オブジェクト
moveDown()　メソッド
間を「.」で**区切って**いる

- **オブジェクト**

  コンピュータでの**操作や処理の対象となるもの**のこと.

  ※ **値が変化するオブジェクト**のことを**変数**と呼んだりもする

- **メソッド**

  オブジェクトに属する**操作や処理**のこと

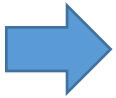
式の抽象化

100 * 1.1

150 * 1.1

400 * 1.1

類似した複数の**式**



a * 1.1

変数 a を使って，複数の**式**を１つにまとめる（**抽象化**）

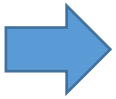
メソッド

100 * 1.1

150 * 1.1

400 * 1.1

類似した複数の**式**



a * 1.1

変数 a を使って，複数の**式**を１つにまとめる（**抽象化**）

```
public class YourClassNameHere {
    public static double foo(double a) {
        return a * 1.1;
    }
    public static void main(String[] args) {
        System.out.printf("%f\n", foo(100));
        System.out.printf("%f\n", foo(150));
        System.out.printf("%f\n", foo(400));
    }
}
```

Print output (dı

```
110.000000
165.000000
440.000000
```

式「**a * 1.1**」を含む**メソッド foo** を**定義**し使用

メソッド

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

- このメソッドの本体は
  「**return a * 1.1;**」

- このメソッドは，式「**a * 1.1**」に，**名前 foo** を付けたものと考えることもできる

式の抽象化とメソッド

抽象化前

```java
public class Main {
    public static void main(String[] args) throws Exception {
        System.out.printf("%f\n", 100 * 1.1);
        System.out.printf("%f\n", 150 * 1.1);
        System.out.printf("%f\n", 400 * 1.1);
    }
}
```

抽象化後

```java
public class Main {
    public static double foo(double a) {
        return a * 1.1;
    }
    public static void main(String[] args) throws Exception {
        System.out.printf("%f\n", foo(100));
        System.out.printf("%f\n", foo(150));
        System.out.printf("%f\n", foo(400));
    }
}
```

類似した複数の**式**

メソッドの定義と使用

出力 入力 コメント 0

```
110.000000
165.000000
440.000000
```

実行結果

出力 入力 コメント 0

```
110.000000
165.000000
440.000000
```

同じ
実行結果になる

まとめ

- **プログラミング**での**オブジェクト**は，**コンピュータでの操作や処理の対象となるもの**のこと
- **メソッド**は，**オブジェクトに属する操作や処理**のこと
- 次の**メソッド**は，**式「 a * 1.1」に，名前 foo を付けたもの**と考えることもできる

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

- **式の抽象化**とは，**変数**を使って，複数の**式**を１つにまとめること

演習

資料：**15 ～ 16**

【トピックス】

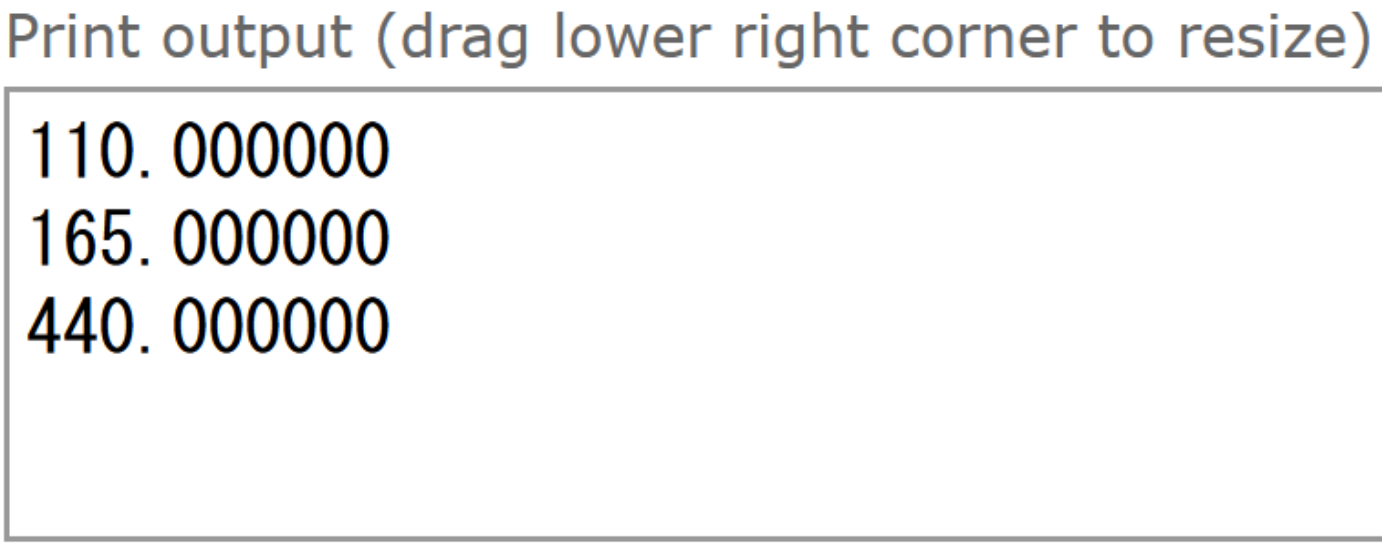
- **式の抽象化**
- **メソッド**

① Java Tutor のエディタで次のプログラムを入れる

Write code in Java 8

```
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          System.out.printf("%f\n", foo(100));
7          System.out.printf("%f\n", foo(150));
8          System.out.printf("%f\n", foo(400));
9      }
10 }
```

② 実行し，結果を確認する

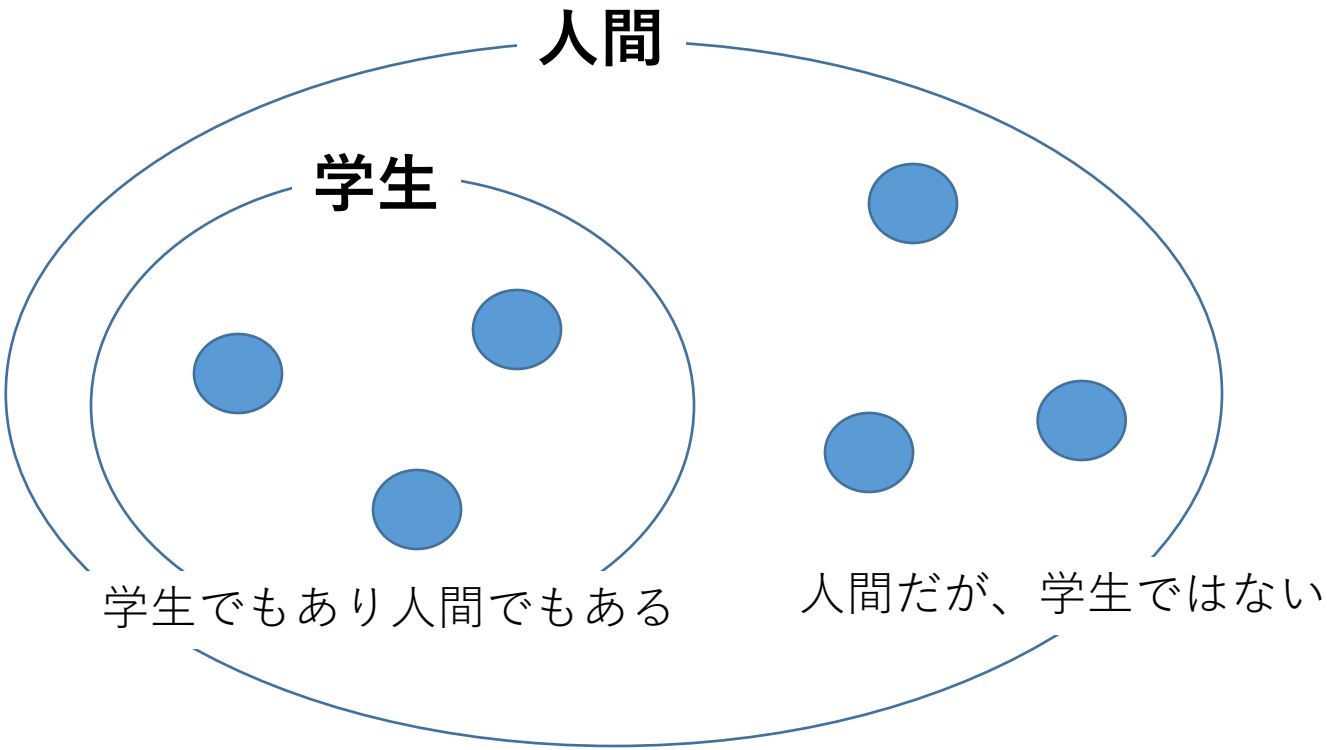


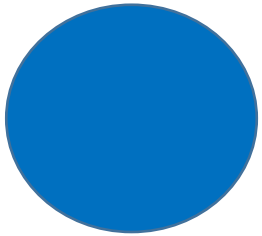
「**Visual Execution**」をクリック．そして「**Last**」をクリック．結果を確認．
「**Edit this code**」をクリックすると，エディタの画面に戻る

13-2. クラス

クラス

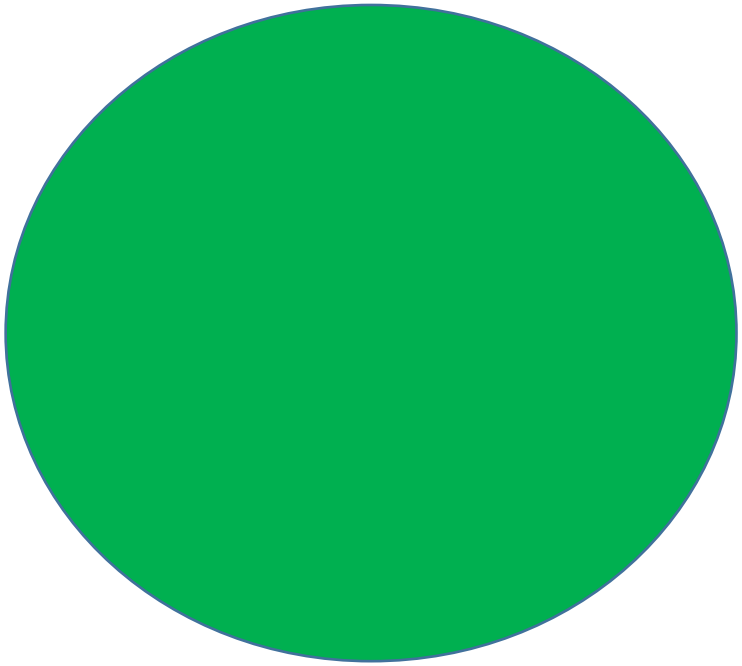
- **クラス**は，**同じ種類のオブジェクトの集まり**と考えることができる





半径 1，場所（8, 10）
色 blue

円（Circle）

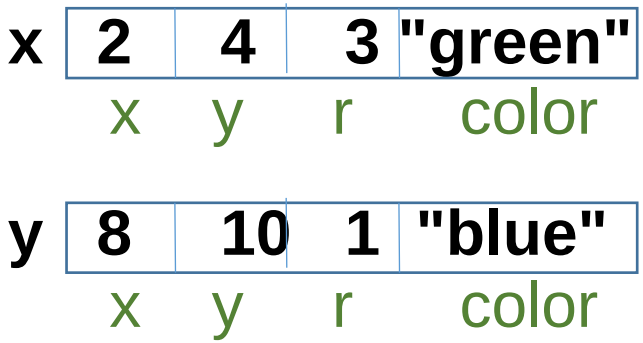


半径 3，場所（2, 4）
色 green

円（Circle）

Java のオブジェクトの生成

次の２つの**オブジェクトを生成する** Java プログラム




```
Circle x = new Circle(2, 4, 3, "green");
Circle y = new Circle(8, 10, 1, "blue");
```

- このとき，次の**クラス**を使うことにする

クラス名 **Circle**

属性 **x, y, r, color**

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

コンストラクタ

クラス定義，コンストラクタ

- Java の**クラス定義**では，**クラス名**，**属性名**と各属性の**データ型**を指定する．メソッド定義も行う．

```
1  class Circle {
2      double x;
3      double y;
4      double r;
5      String color;
6      public Circle(double x, double y, double r, String color) {
7        this.x = x;
8        this.y = y;
9        this.r = r;
.0       this.color = color;
.1     }
.2 }
```

- **コンストラクタ**とは，**オブジェクト**の生成を行う**メソッド**のことである．

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

まとめ

- Java の**クラス定義**では，**クラス名**，属性名と各属性の**データ型**を指定する.
- **コンストラクタ**とは，**オブジェクト**の生成を行う**メソッド**のことである.

- キーワード

class　　**クラス定義**

new　　**コンストラクタ**の呼び出し

演習

資料：**24 ～ 25**

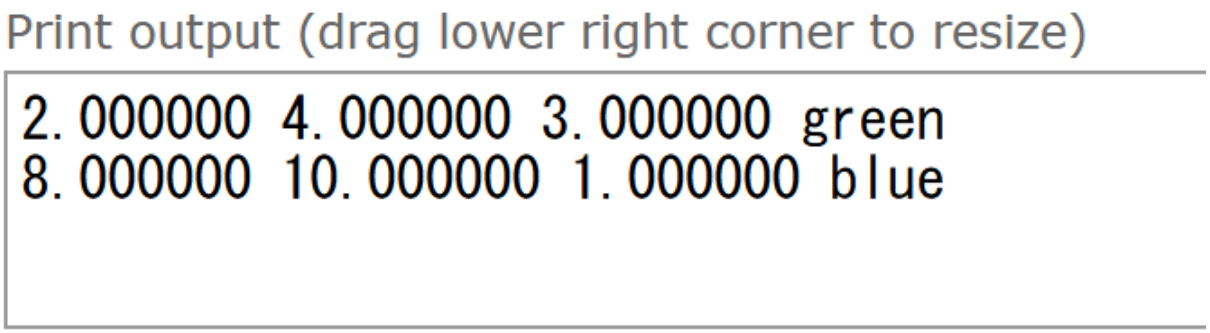
【トピックス】

- **クラス**
- **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


```
class Circle {
    double x;
    double y;
    double r;
    String color;
    public Circle(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.printf("%f %f %f %s¥n", this.x, this.y, this.r, this.color);
    }
}
public class YourClassNameHere {
    public static void main(String[] args) {
        Circle x = new Circle(2, 4, 3, "green");
        Circle y = new Circle(8, 10, 1, "blue");
        x.printout();
        y.printout();
    }
}
```

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

メソッドとクラス

- プログラミングでの**メソッド**とは，**オブジェクトに関する操作や処理**のこと
- **メソッド**は，**クラス**に属する
- **メソッド**内のプログラムは，その**メソッド**が所属する**クラス**の**属性**や**メソッド**への**アクセス権がある**

属性やメソッドのアクセス

- **「オブジェクト名」＋「.」で属性やメソッドにアクセス**する
- **メソッド内**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 ＋「.」**

属性アクセス

x	2	4	3	"green"
	x	y	r	color

y	8	10	1	"blue"
	x	y	r	color

「オブジェクト名」＋「.」で属性やメソッドにアクセスする

```
public class YourClassNameHere {
    public static void main(String[] args) {
        Circle x = new Circle(2, 4, 3, "green");
        Circle y = new Circle(8, 10, 1, "blue");
        System.out.printf("%f\n", x.x);
        System.out.printf("%s\n", y.color);
    }
}
```

Print output (drag lower right corner to resize)

```
2.000000
blue
```

メソッド内での属性アクセス

メソッド内で，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 ＋「.」**

※「**this**」は，「メソッドが処理中のオブジェクトのことである」とみなすことも.

```
1  class Circle {
2      double x;
3      double y;
4      double r;
5      String color;
6      public Circle(double x, double y, double r, String color) {
7          this.x = x;
8          this.y = y;
9          this.r = r;
10         this.color = color;
11     }
```

まとめ

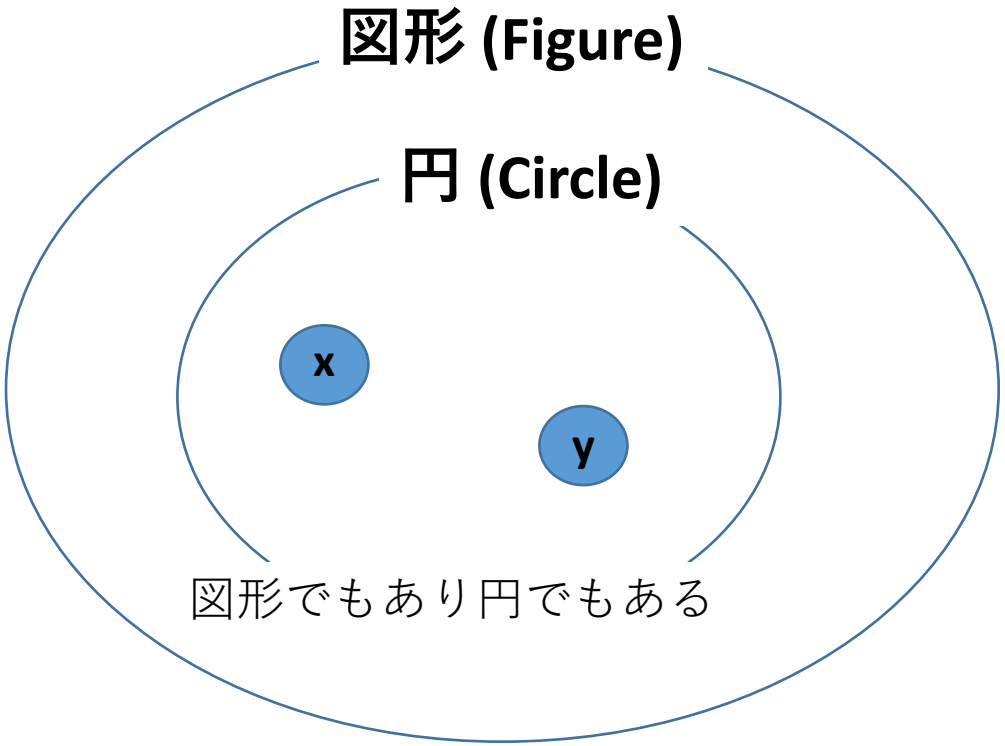
- **メソッド**は，**クラス**に属する
- **「オブジェクト名」＋「.」**で**属性**や**メソッド**に**アクセス**する
- **メソッド**内のプログラムは，その**メソッド**が所属する**クラス**の**属性**や**メソッド**への**アクセス権がある**
- **メソッド内**で，その**メソッドが所属するクラス**で定義された**属性**や**メソッド**にアクセスするときは**this ＋「.」**

13-3. スーパークラス，サブクラス，継承

スーパークラス，サブクラス

- **スーパークラス**「**図形 (Figure)**」
- **サブクラス**「**円 (Circle)**」

「**円 (Circle)**」の**オブジェクト**は，すべて「**図形 (Figure)**」である



継承の例

- **継承**とは，**スーパークラス**の**属性**と**メソッド**を**サブクラス**が**受け継ぐ**こと

次のように考える

- **図形 (Figure)** の**属性**　→ すべて，**円 (Circle)** に**継承**される

  x, y, color

- **円 (Circle)** にしかない**属性**

  r

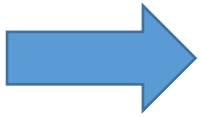
⇒ **円の属性は x, y, color, r**

クラスの類似性

Figure
属性

x
y
color

スーパークラス



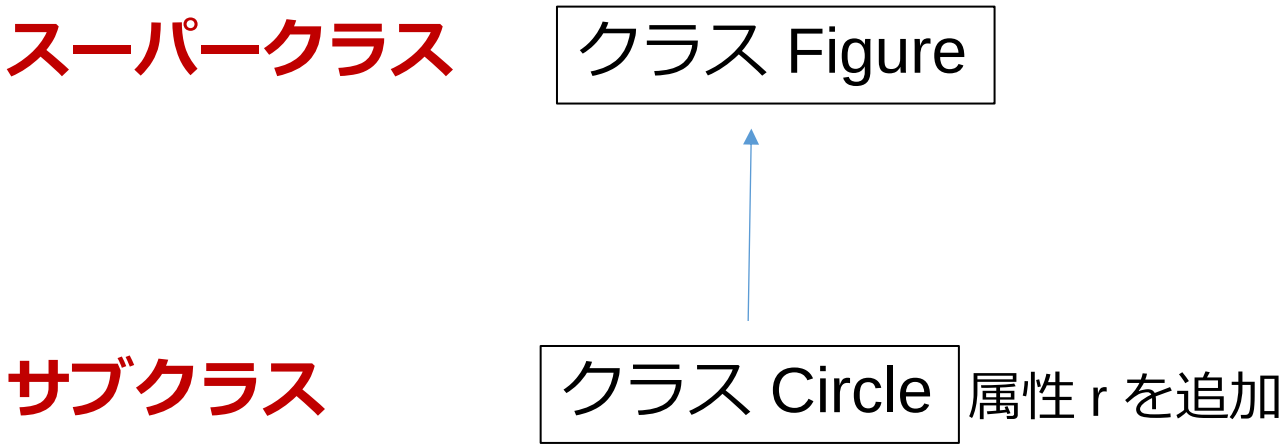
Circle
属性

x
y
color
r　　追加

サブクラス

スーパークラス，サブクラス

- **サブクラス**は，スーパークラスの**属性**と**メソッド**を**すべて持つ**
- **サブクラス**で，スーパークラスにない**属性**や**メソッド**が**追加されることがある**



クラス Figure の定義

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

クラス名 **Figure**

属性 **x, y, color**

Java でのクラスの親子関係の書き方

親子関係の指定「**class Circle extends Figure**」

子クラスである Circle で**追加される属性，メソッド**を書く

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

コンストラクタの定義.
super(x, y, color) により，**親クラス**の**コンストラクタ**を呼び出していることに注意

スーパークラス、サブクラスのためのキーワード

キーワード	
extends	**スーパークラス**の指定
super	**スーパークラス**の**コンストラクタ**の呼び出し

演習

資料：**40 ～ 42**

【トピックス】

- **スーパークラス，サブクラス**
- **継承**

① Java Tutor のエディタで次のプログラムを入れる


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

クラス定義

```
class Circle extends Figure {
    double r;
    public Circle(double x, double y, double r, String color) {
        super(x, y, color);
        this.r = r;
    }
    public void printout() {
        System.out.printf("%f %f %f %s¥n", this.x, this.y, this.r, this.color);
    }
}
```

クラス定義

次のページに続く

次のソースコードを入れる

```
public class YourClassNameHere {
    public static void main(String[] args) {
        Circle x = new Circle(2, 4, 3, "green");
        Circle y = new Circle(8, 10, 1, "blue");
        x.printout();
        y.printout();
    }
}
```

② 実行し，結果を確認する

Print output (drag lower right corner to resize)

```
2.000000 4.000000 3.000000 green
8.000000 10.000000 1.000000 blue
```

「**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

- **クラス階層**とは，**複数のクラスが親子関係をなす**こと
- クラス①が**親**，クラス②が**子**であるとき
  - クラス②は，クラス①の**属性**と**メソッド**を**すべて持つ**
  - クラス②で，クラス①にない**属性**や**メソッド**が**追加されることがある**
- **親子関係**の指定は，「**class Circle extends Figure**」のように書く．Circle が子，Figure が親.
- **継承**とは，**親クラス**の**属性**と**メソッド**を**子クラス**が**受け継ぐ**こと
- **親クラス**のことを「**スーパークラス**」，**子クラス**のことを「**サブクラス**」ともいう

2つのクラスのプログラム（親子関係にしない場合）

Ball

```
class Ball {
  double x;
  double y;
  String color;
  public Ball(double x, double y, String color) {
    this.x = x;
    this.y = y;
    this.color = color;
  }
  public void move(double xx, double yy) {
    this.x = this.x + xx;
    this.y = this.y + yy;
  }
  public void reset() {
    this.x = x;
    this.y = y;
  }
}
```

Circle

```
class Circle {
  double x;
  double y;
  String color;
  double r;
  public Circle(double x, double y, String color, double r) {
    this.x = x;
    this.y = y;
    this.color = color;
    this.r = r;
  }
  public void move(double xx, double yy) {
    this.x = this.x + xx;
    this.y = this.y + yy;
  }
  public void reset() {
    this.x = x;
    this.y = y;
  }
}
```

r の部分が違う

全く同じ

2つのクラスのプログラム
親子関係にしない場合とする場合の比較

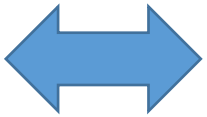
Ball

```
class Ball {
  double x;
  double y;
  String color;
  public Ball(double x, double y, String color) {
    this.x = x;
    this.y = y;
    this.color = color;
  }
  public void move(double xx, double yy) {
    this.x = this.x + xx;
    this.y = this.y + yy;
  }
  public void reset() {
    this.x = x;
    this.y = y;
  }
}
```

Circle

```
class Circle {
  double x;
  double y;
  String color;
  double r;
  public Circle(double x, double y, String color, double r) {
    this.x = x;
    this.y = y;
    this.color = color;
    this.r = r;
  }
  public void move(double xx, double yy) {
    this.x = this.x + xx;
    this.y = this.y + yy;
  }
  public void reset() {
    this.x = x;
    this.y = y;
  }
}
```

働きは同じ



Ball

```
class Ball {
  double x;
  double y;
  String color;
  public Ball(double x, double y, String color) {
    this.x = x;
    this.y = y;
    this.color = color;
  }
  public void move(double xx, double yy) {
    this.x = this.x + xx;
    this.y = this.y + yy;
  }
  public void reset() {
    this.x = x;
    this.y = y;
  }
}
```

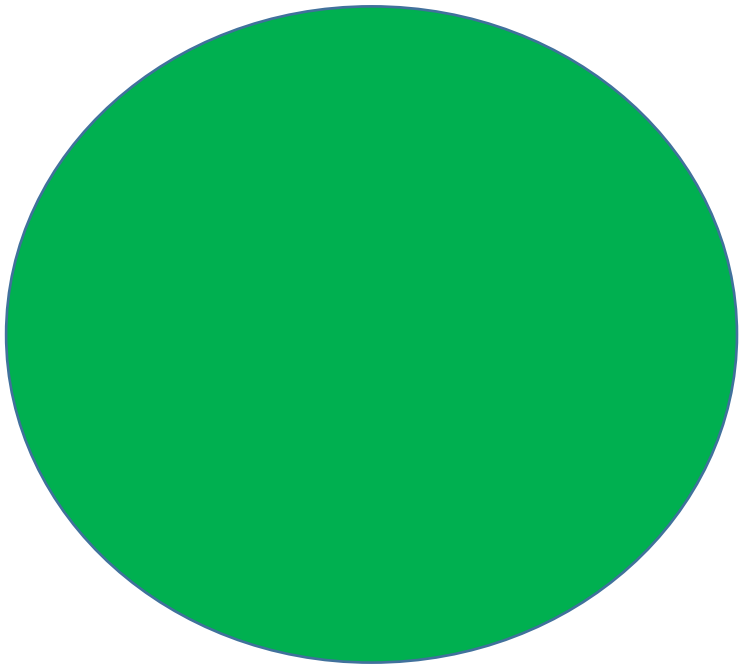
Circle

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

親子関係にしない
（同じようなプログラムを繰り返す）

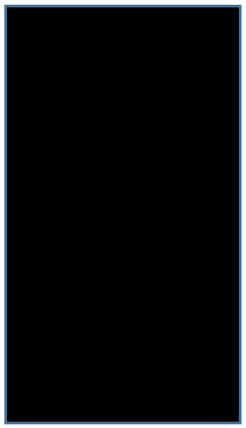
親子関係にする

13-4. クラスの抽象化



半径 3，場所（2, 4）
色 green

円（Circle）



幅 1, 高さ 2, 場所（6, 4）
色 black

長方形
(Rectangle)

クラスの類似性

- 類似した２つの**クラス**

円 (Circle)

属性

x

y

color

r　　半径

x, y, color は
共通

長方形 (Rectangle)

属性

x

y

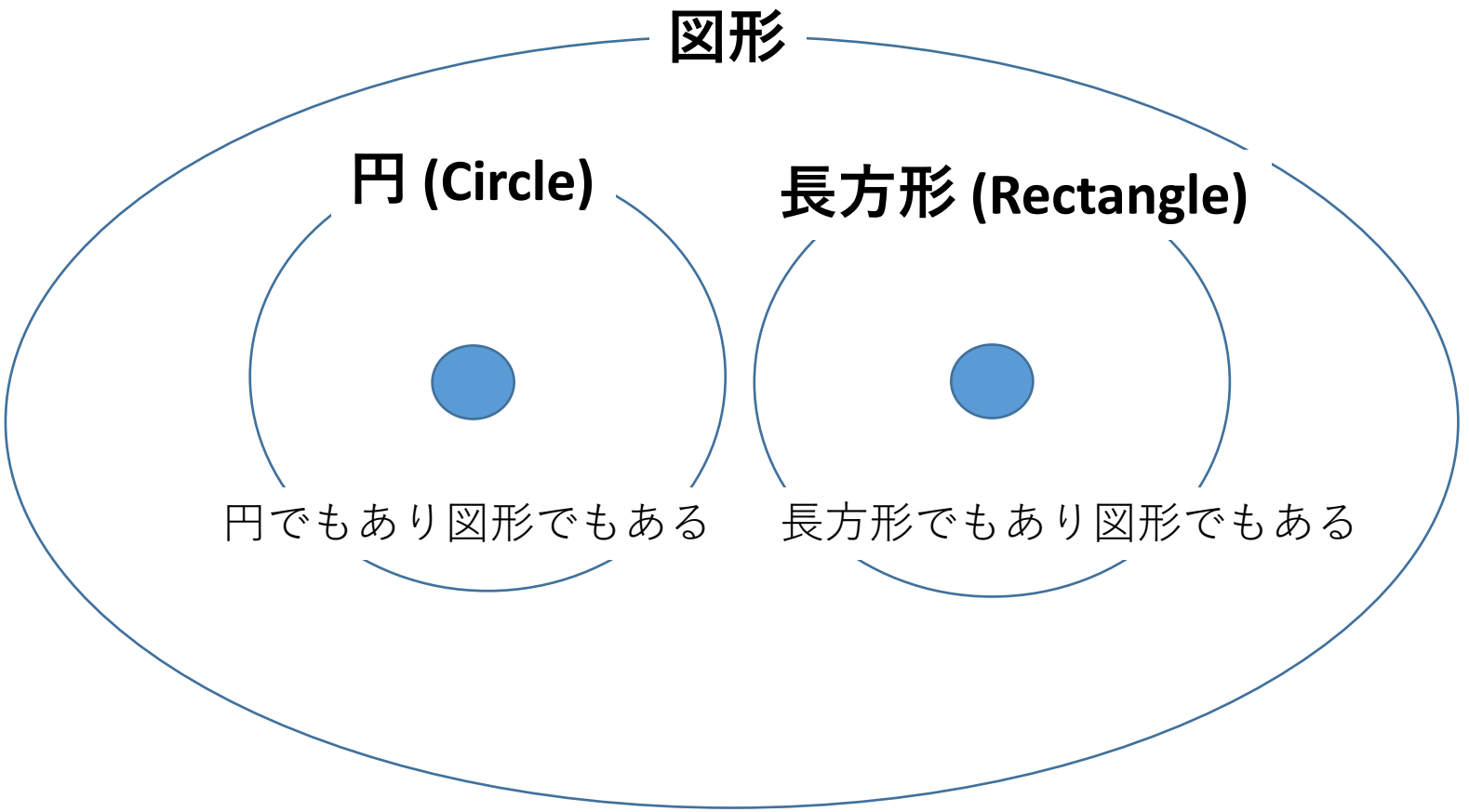
color

width　幅

height　高さ

クラス

- **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



クラスの抽象化

円 (Circle)

属性

- x
- y
- color
- **r　半径**

長方形 (Rectangle)

属性

- x
- y
- color
- **width　幅**
- **height　高さ**

図形 (Figure)

属性

- x
- y
- color

共通属性を持つ

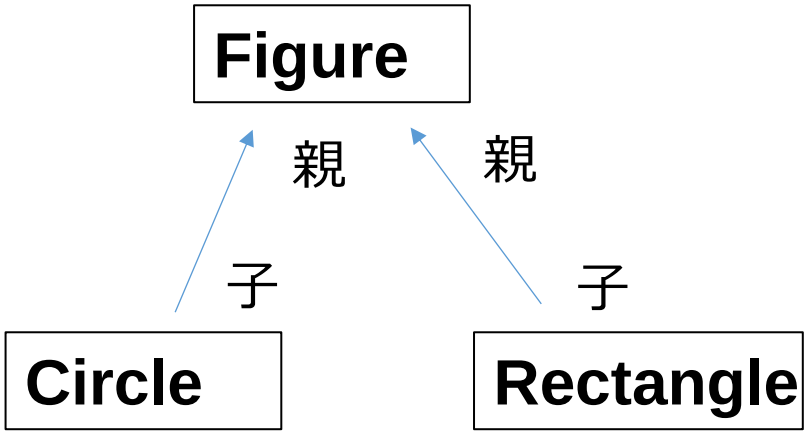
Java のオブジェクトの生成

次の２つの**オブジェクトを生成する** Java プログラム


```
Circle x = new Circle(2, 4, 3, "green");
Rectangle a = new Rectangle(6, 4, 1, 2, "blue");
```

クラス階層は何のため？

- **似通ったクラス Circle, Rectangle** を使いたい．**プログラムのミスを減らす**ため
- 将来，図形の種類を増やすときにも有効



クラス Circle, クラス Rectangle が似ている．

共通する機能を、スーパークラス Figure にまとめる．

演習

資料：**54 ～ 56**

【トピックス】

・クラスの抽象化

① Java Tutor のエディタで次のプログラムを入れる


```
class Figure {
    double x;
    double y;
    String color;
    public Figure(double x, double y, String color) {
        this.x = x;
        this.y = y;
        this.color = color;
    }
}
class Circle extends Figure {
    double r;
    public Circle(double x, double y, double r, String color) {
        super(x, y, color);
        this.r = r;
    }
    public void printout() {
        System.out.printf("%f %f %f %s¥n", this.x, this.y, this.r, this.color);
    }
}
```

ここまでは
前のプログラムそのまま

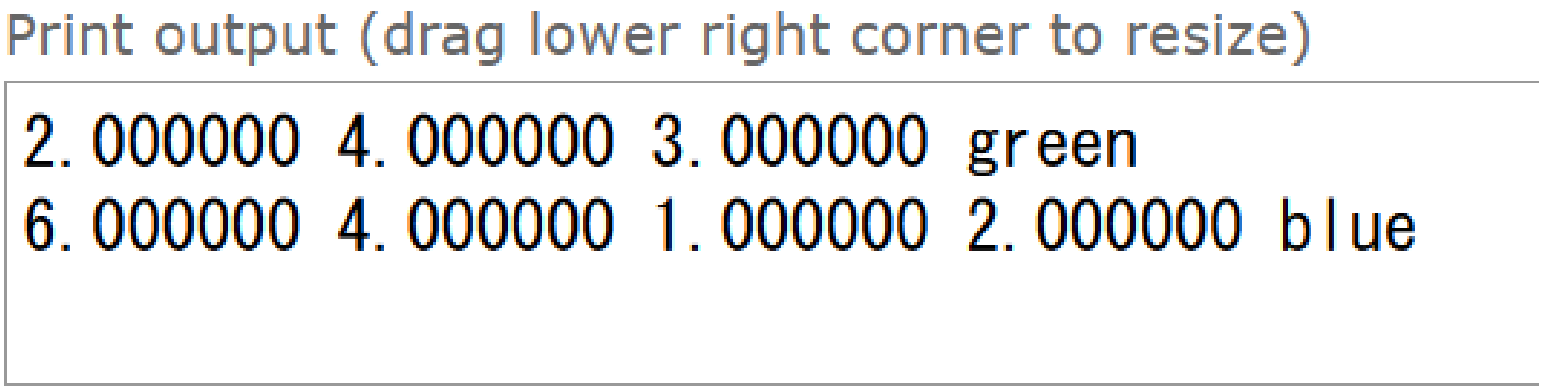
次のページに続く

続き

```
class Rectangle extends Figure {
    double width;
    double height;
    public Rectangle(double x, double y, double w, double h, String color) {
        super(x, y, color);
        this.width = w;
        this.height = h;
    }
    public void printout() {
        System.out.printf("%f %f %f %f %s", this.x, this.y, this.width, this.height, this.color);
    }
}
public class YourClassNameHere {
    public static void main(String[] args) {
        Circle x = new Circle(2, 4, 3, "green");
        Rectangle a = new Rectangle(6, 4, 1, 2, "blue");
        x.printout();
        a.printout();
    }
}
```

クラス定義

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

13-5. Java プログラム例

配列と繰り返し

```
1  public class YourClassNameHere {
2      public static void main(String[] args) {
3          double x[] = {8, 6, 4, 2, 3};
4          double y[] = {0, 0, 0, 0, 0};
5          int i;
6          for(i=0; i<=4; i++) {
7              y[i] = x[i] * 1.1;
8          }
9          for(i=0; i<=4; i++) {
10             System.out.println(y[i]);
11         }
12     }
13 }
```

配列の組み立て

「y[i] = x[i] * 1.1」を i の値を変えながら5回繰り返す

演習

資料：**60 ～ 62**

【トピックス】

- **配列**
- **繰り返し**

① 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


```
public class YourClassNameHere {
   public static void main(String[] args) {
      double x[] = {8, 6, 4, 2, 3};
      double y[] = {0, 0, 0, 0, 0};
      int i;
      for(i=0; i<=4; i++) {
         y[i] = x[i] * 1.1;
      }
      for(i=0; i<=4; i++) {
         System.out.println(y[i]);
      }
   }
}
```

Print output (drag lower right corner to resize)

```
8. 8
6. 6000000000000005
4. 4
2. 2
3. 3000000000000003
```

疑似乱数を１０個作る

```
import java.util.Random;
```

標準ライブラリ
java.util.Random のインポート

```
public class YourClassNameHere {
    public static void main(String[] args) {
        Random r = new Random();
        int i, a;
        for(i=0; i<10; i++) {
            a = r.nextInt(100);
            System.out.println(a);
        }
    }
}
```

疑似乱数の生成と
表示を１０回
繰り返し

② 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


```
import java.util.Random;

public class YourClassNameHere {
    public static void main(String[] args) {
        Random r = new Random();
        int i, a;
        for(i=0; i<10; i++) {
            a = r.nextInt(100);
            System.out.println(a);
        }
    }
}
```

Print output (drag lower right corner to resize)

```
54
3
37
32
33
```

表示を確認

0 から 99 の乱数が 10個表示される.

関連ページ

- **Java プログラミング入門**

GDB online を使用

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

- **Java の基本**

Java Tutor, GDB online を使用

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

- **Java プログラム例**

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

13-1

```
public class YourClassNameHere {
    public static double foo(double a) {
        return a * 1.1;
    }
    public static void main(String[] args) {
        System.out.printf("%f¥n", foo(100));
        System.out.printf("%f¥n", foo(150));
        System.out.printf("%f¥n", foo(400));
    }
}
```

13-2

```
class Circle {
    double x;
    double y;
    double r;
    String color;
    public Circle(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.printf("%f %f %f %s¥n", this.x, this.y, this.r, this.color);
    }
}
public class YourClassNameHere {
    public static void main(String[] args) {
        Circle x = new Circle(2, 4, 3, "green");
        Circle y = new Circle(8, 10, 1, "blue");
        x.printout();
        y.printout();
    }
}
```

13-3

```
class Figure {
    double x;
    double y;
    String color;
    public Figure(double x, double y, String color) {
        this.x = x;
        this.y = y;
        this.color = color;
    }
}
class Circle extends Figure {
    double r;
    public Circle(double x, double y, double r, String color) {
        super(x, y, color);
        this.r = r;
    }
    public void printout() {
        System.out.printf("%f %f %f %s¥n", this.x, this.y, this.r, this.color);
    }
}
public class YourClassNameHere {
    public static void main(String[] args) {
        Circle x = new Circle(2, 4, 3, "green");
        Circle y = new Circle(8, 10, 1, "blue");
        x.printout();
        y.printout();
    }
}
```

13-4

```
class Figure {
    double x;
    double y;
    String color;
    public Figure(double x, double y, String color) {
        this.x = x;
        this.y = y;
        this.color = color;
    }
}
class Circle extends Figure {
    double r;
    public Circle(double x, double y, double r, String color) {
        super(x, y, color);
        this.r = r;
    }
    public void printout() {
        System.out.printf("%f %f %f %s¥n", this.x, this.y, this.r, this.color);
    }
}
class Rectangle extends Figure {
    double width;
    double height;
    public Rectangle(double x, double y, double w, double h, String color) {
        super(x, y, color);
        this.width = w;
        this.height = h;
    }
    public void printout() {
        System.out.printf("%f %f %f %f %s", this.x, this.y, this.width, this.height, this.color);
    }
}

public class YourClassNameHere {
    public static void main(String[] args) {
        Circle x = new Circle(2, 4, 3, "green");
        Rectangle a = new Rectangle(6, 4, 1, 2, "blue");
        x.printout();
        a.printout();
    }
}
```

13-5

```
public class YourClassNameHere {
    public static void main(String[] args) {
        double x[] = {8, 6, 4, 2, 3};
        double y[] = {0, 0, 0, 0, 0};
        int i;
        for(i=0; i<=4; i++) {
            y[i] = x[i] * 1.1;
        }
        for(i=0; i<=4; i++) {
            System.out.println(y[i]);
        }
    }
}
```

13-5 の 2つめ

```
import java.util.Random;

public class YourClassNameHere {
    public static void main(String[] args) {
        Random r = new Random();
        int i, a;
        for(i=0; i<10; i++) {
            a = r.nextInt(100);
            System.out.println(a);
        }
    }
}
```