# Java の基礎まとめ

（Java プログラミング体験学習）

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

金子邦彦

# アウトライン

1 オブジェクト，メソッド

2 クラス定義，コンストラクタ

3 クラス階層，継承

- 重要トピックスの説明をまとめている
- 体験学習は https://www.kkaneko.jp/cc/jp/index.html

# 1. オブジェクト，メソッド

# オブジェクト

- **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.
- **オブジェクト**は１つあるいは複数の**データ**を持つことが**できる**

# メソッド

- **メソッド**は，**オブジェクトに属する操作や処理**のこと

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

- **引数（ひきすう）**とは，**メソッドに渡す値**のこと

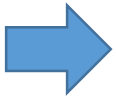
# 式の抽象化

**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**）

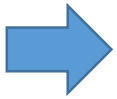
# メソッド

**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));
    }
}
```

**メソッド foo** を使用.
100, 150, 400 は**引数**

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

Print output (d

```
110.000000
165.000000
440.000000
```

# 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

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

類似した複数の**式**

Print output (drag

```
110.000000
165.000000
440.000000
```

実行結果

## 抽象化後

```
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));
    }
}
```

**メソッド**の定義と使用

Print output (dı

```
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

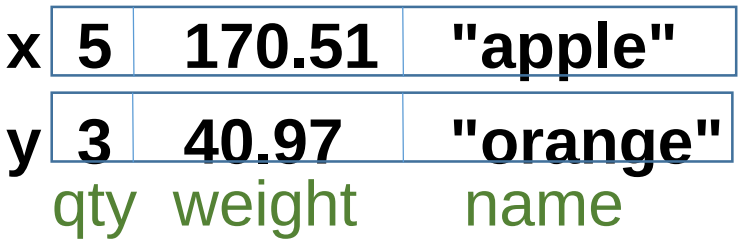
```
public static double foo(double a) {
  return a * 1.1;
}
```

- **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

# 2. クラス定義，コンストラクタ

# 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




- クラス Ball のオブジェクト生成を行うプログラム

```
C x = new C(5, 170.51, "apple");
C y = new C(3, 40.97, "orange");
```

# クラス定義

**クラス定義**の中には，**属性**の定義（**属性名**と**データ型**），**コンストラクタ**の定義，その他**メソッド**の定義を含める.

```
1  class C {
2    int qty;
3    double weight;
4    String name;
5    public C(int qty, double weight, String name) {
6        this.qty = qty;
7        this.weight = weight;
8        this.name = name;
9    }
10 }
```

**オブジェクト**の生成を行う**メソッド**のことを**コンストラクタ**という

# 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

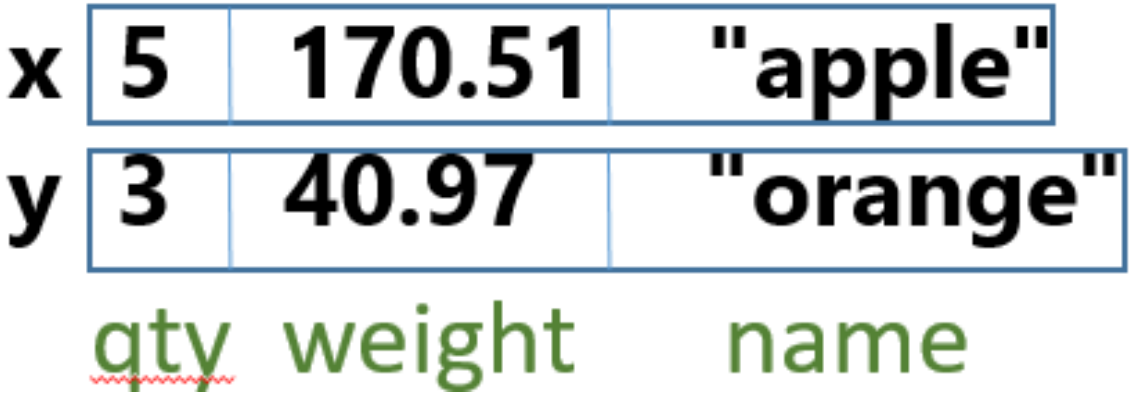
# メソッドとクラス

- 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




**「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) {
        C x = new C(5, 170.51, "apple");
        C y = new C(3, 40.97, "orange");
        System.out.printf("%d", x.qty);
    }
}
```

# 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.

```
class C {
  int qty;
  double weight;
  String name;
  public C(int qty, double weight, String name) {
    this.qty = qty;
    this.weight = weight;
    this.name = name;
  }
}
```

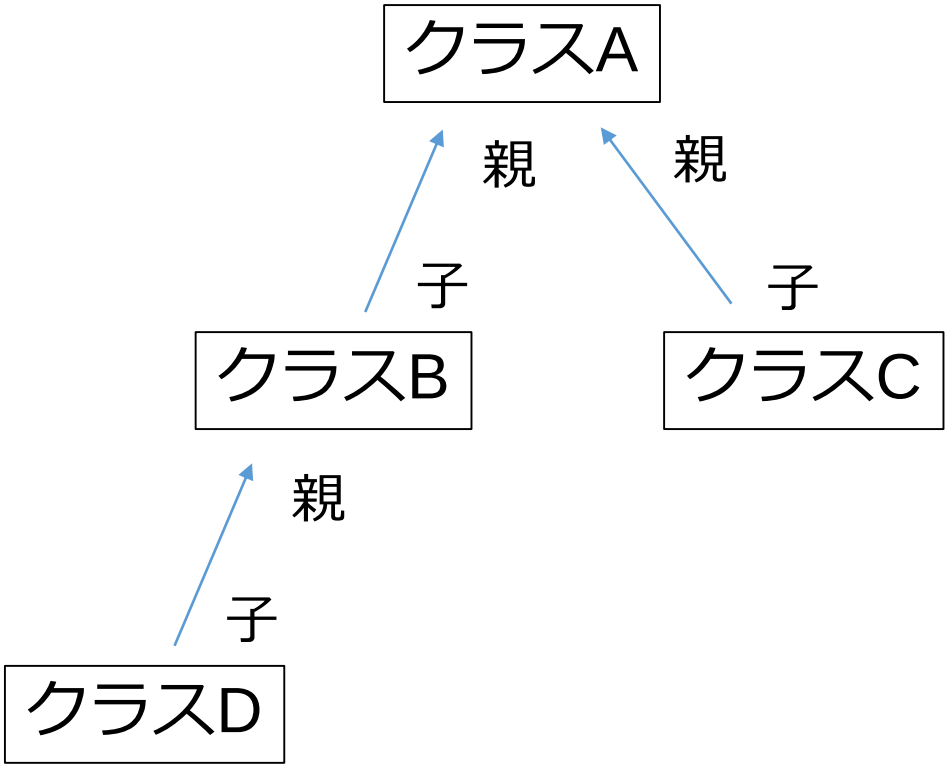
# まとめ

- **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 ＋「.」**

# 3. クラス階層，継承

# クラス階層

**クラス階層**とは，**複数のクラスが親子関係をなす**こと

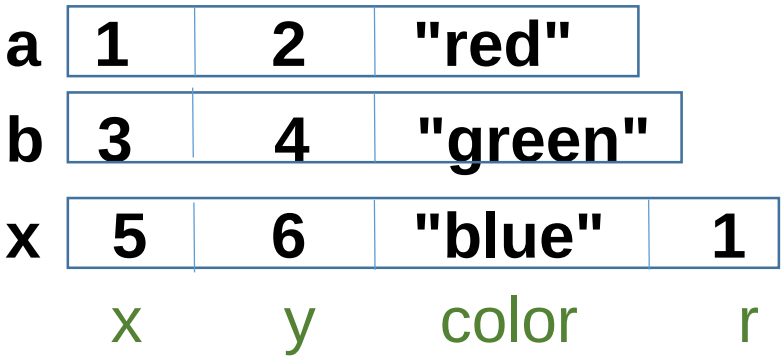


# 継承

- **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

# Python のオブジェクトの生成

- 次の３つの**オブジェクト**を生成




- オブジェクト生成を行うプログラム

```
Ball a = new Ball(1, 2, "red");
Ball b = new Ball(2, 4, "green");
Circle x = new Circle(5, 6, "blue", 1);
```

# クラスの類似性

- 類似した２つの**クラス**

Ball

属性

x

y

color

メソッド

move

reset

x, y, color は**同じ**

**r の有り無しが違う**

メソッドの**名前も中身も全く同じ**とする

Circle

属性

x

y

color

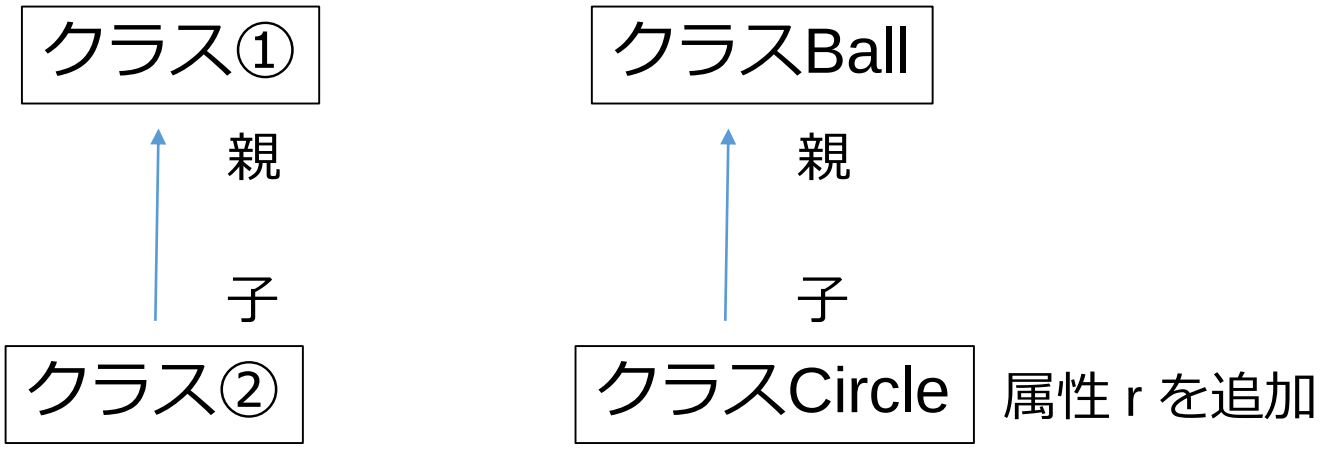
r

move

reset

# クラスの親子関係

- 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**



# クラスの親子関係

```
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;
  }
}
```

クラス名 **Ball**

属性 **x, y, color**

メソッド **move, reset**

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

クラス名 **Circle**

属性 **x**, **y**, **color, r**

メソッド **move, reset**

クラス **Circle** は，**親クラス**であるクラス **Ball** の**属性**と**メソッド**を**継承**する.

# 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 Ball**」

**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

# Java の子クラスでのコンストラクタ

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

**親クラス**の**コンストラクタ**を呼び出して，**属性値**を設定

**子クラス**で追加した**属性**を設定

# 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

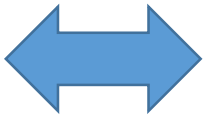
```
class Ball {
  double x;
  double y;
  String color;
  public Ball(double x, double y, String color) {
    this.x = x;
    this.y = y;
    this.color = color;
  }
  public void move(double xx, double yy) {
    this.x = this.x + xx;
    this.y = this.y + yy;
  }
  public void reset() {
    this.x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

**親子関係にしない**
（同じようなプログラムを繰り返す）

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;
  }
}
```

**親子関係にする**

# 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 Ball**」のように書く．Circle が子，Ball 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