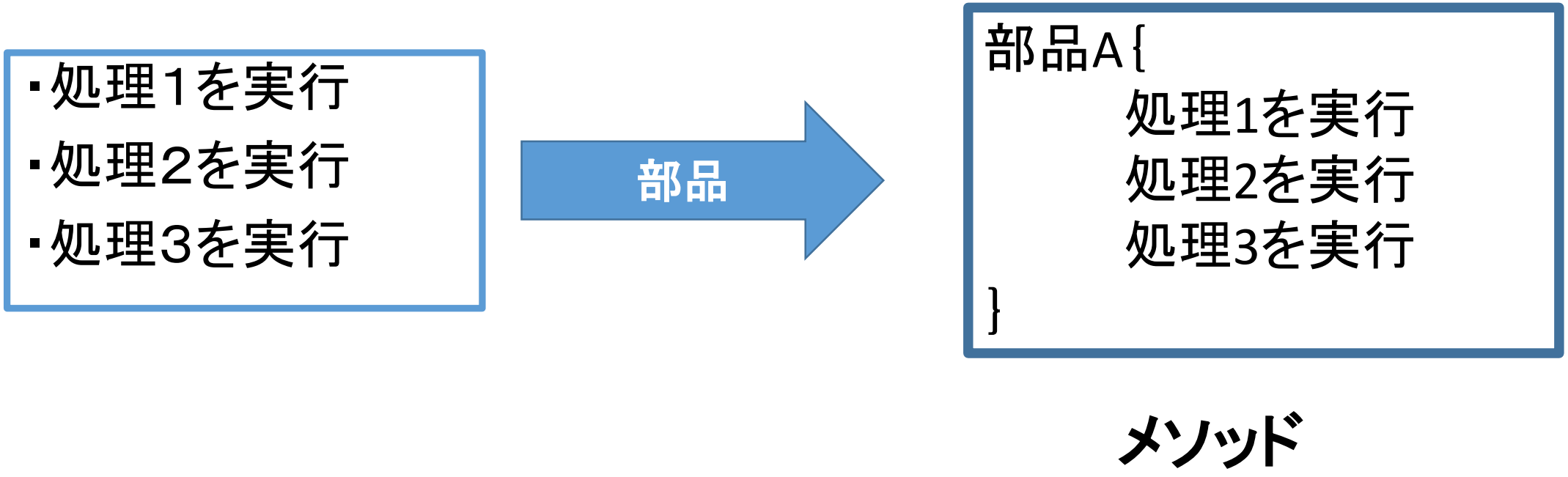
Java講習　第9回

～クラス　その1.1～

復習



復習

料理を作る{
　　食料を集める
　　調理する
　　盛り付け
}

料理の片付け{
　　皿を集める
　　洗う
　　乾燥させる
}

朝食{
　　・「料理を作る」
　　・食べる
　　・「料理の片付け」
}

復習

朝食{
- 食料を集める
- 調理する
- 盛り付け
- 食べる
- 皿を集める
- 洗う
- 乾燥させる

}

朝食{
- 「料理を作る」
- 食べる
- 「料理の片付け」

}

見やすい

朝食{

- 「料理を作る」
- 食べる
- 「料理の片付け」

}

```
public class Breakfast{
    public static void main(String[] args){
        cook();
        //食べる
        clean();
    }

    void cook(){
        //食料を集めて調理する
        //盛り付け
    }

    void clean(){
    //皿を集めて洗って乾燥
    }
}
```

```
部品A{
        ・処理1を実行
        ・処理2を実行
        部品Bを実行
}
```

```
設計図{
        部品(){
                //処理
        }
        部品2(){
                //処理
        }
}
```

あくまで部品なのでまとめる必要がある！

```
部品A{
        ・処理1を実行
        ・処理2を実行
        部品Bを実行
}
```

```
設計図{
        部品(){
                //処理
        }
        部品2(){
                //処理
        }
}
```

あくまで部品なのでまとめる必要がある！

```
設計図{
        ・状態1
        ・状態2
        部品(){
                //処理
        }
        部品2(){
                //処理
        }
}
```

設計図には状態（値）を保持できる

```
設計図{
        ・状態1
        ・状態2
        部品(){
                //処理
        }
        部品2(){
                //処理
        }
}
```

```
人間{
        ・健康状態
        ・身長
        歩く(){
                //処理
        }
        食事する(){
                //処理
        }
}
```

設計図・・・クラス

人間{
・健康状態
・身長
歩く(){
//処理
}
食事する(){
//処理
}
}

```
public class Human{
        String  info;
        double  tall;
        public static void main(String[] args){
                walk();         //メソッド呼ぶ
        }

        void walk(){
                //歩く
        }

        void eat(String food){
                //食事する
        }
}
```

クラスとは

- プログラムの一番外側に記述されていたもの

```
public class Main{
        public static void main(String[] args){
                //処理
        }
}
```

クラスとは

- プログラムの一番外側に記述されていたもの

```
public class Main{
        public static void main(String[] args){
                //処理
        }
}
```

クラスとは

- メソッドと変数の集まり

```
public class Human{
        String  info;
        double  tall;
        public static void main(String[] args){
                walk();
        }
        void walk(){
        }
}
```

クラスとは

- テンプレートのような雛型，設計書，定義　　⇒複製可能

```
public class Human{
        String  info;
        double  tall;
        public static void main(String[] args){
                walk();
        }
        void walk(){
        }
}
```

値を変えると別の種類ができる
ただし、骨格は同じ

```
public△class△Car{
        int   speed =40;
        int   size = 300;
        String   color;
        void   accel(){
        }
        void   brake(){
        }
}
```

```
public△class△Car{
        int   speed =120;
        int   size = 100;
        String   color;
        void   accel(){
        }
        void   brake(){
        }
}
```

クラスとは

- メソッド….. 処理をまとめる。 部品
- クラス….. メソッドをまとめる, 状態を保持する。設計図

```
部品A{
        処理1を実行
        処理2を実行
        処理3を実行
}
```

```
設計図{
        部品{
                //処理
        }
}
```

クラス（設計図）のつくり方

[修飾子]△class△クラス名{

}

class　クラス名{

}

※修飾子

public　private　などがある

public class クラス名{

}

クラス（設計図）のつくり方

Main.java

```
public class Main{
        public static void main(){
        }
}
```

※ただし、1つのファイルにpublicのクラスは1つ

public のクラス名とファイル名は同じ

クラス名の最初は大文字！のほうがよい

```
public class Main{
        public static void main(){
        }
}
class Sub{
        //状態(変数)や部品(メソッド)
}
```

```
public 設計図{
        最初に実行する部品(){
                //処理
        }
        部品2(){
                //処理
        }
}

設計図2{
        部品A(){
        }
}
```

クラスはあくまで　設計図/テンプレート

中身が詰まってない(実体がない)

操作ができない

中身を詰める必要がある。

中身を詰めることを**インスタンスを生成する**という

インスタンス生成方法 (クラスの使い方)

- new 演算子を使う

クラス名△変数 = new△クラス名();

インスタンス生成

呼ぶクラス専用の
型を用意

インスタンス
を変数で保持

変数宣言

```
int  変数1;
変数1 = 10;

String 変数2;
変数2 = “Hello”

クラス名  変数;
変数 = new クラス名();
```

インスタンス生成方法 (クラスの使い方)

クラス名△変数　= new△クラス名();

例:

Subクラスを生成する場合

Sub△sub1 = new△Sub();

Sub△sub2 = new△Sub();

クラス使用例

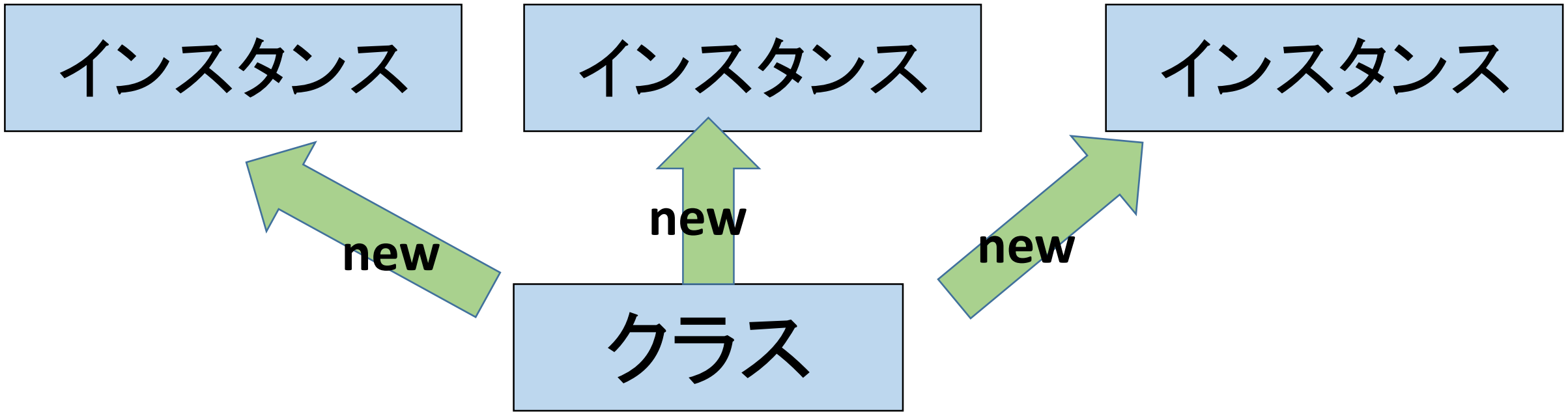
```
public class Main{
        public static void main(String[] args){
                Sub sub1 = new Sub();
        }
}
class Sub{
        void method1(){
        }
}
```

Sub クラス

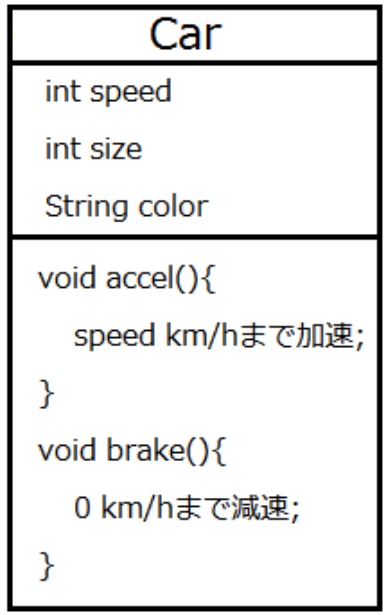
インスタンス生成すると…

- **インスタンスを生成したクラスのメソッド, 変数が使える!**
  →設計図から本体の機能が使える!

クラスを元に作られた操作できる部品



クラス

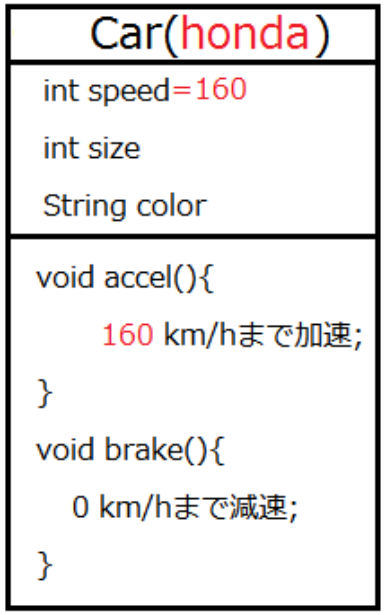


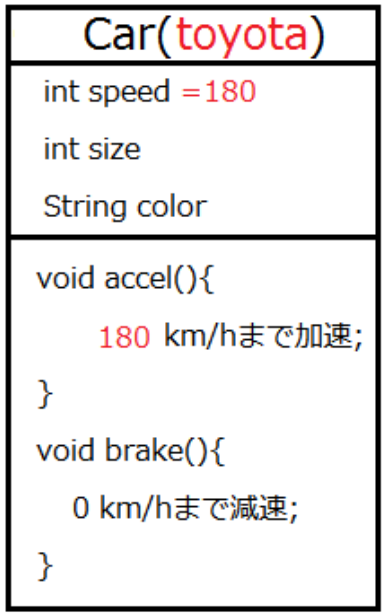
インスタンス生成

```
Car honda=new Car();
honda.speed=160;
```

インスタンス生成

```
Car toyota=new Car();
toyota.speed=180;
```





インスタンス生成後

インスタンス生成した
クラスの変数・メソッドの使い方

- **変数を使う**

  インスタンス変数.クラス内の変数

※メソッド内の変数は呼べない

- **メソッドを使う**

  インスタンス変数.メソッド名();

```
Sub   sub1 = new Sub();
System.out.println(sub1.a);
sub1.a = 10;


sub1.printA();
```

```
public class Main{
        int a;
        public static void main(String[] args){
                System.out.println(a);
                method1();

                Sub sub1  = new Sub();
                System.out.println(sub1.subA);
                sub1.subMethod();
        }
        void method1(){ ... }
}
```

フィールド変数

```
class Sub{
        int subA=5;
        void subMethod(){
        }
}
```

フィールド変数

例題:

1.mainメソッドのあるクラスと別にHumanクラスを作成

Humanクラス
フィールド変数
・String型で変数名　food

メソッド
・eatメソッド
引数: String型1つ
処理: 変数foodに代入する

2.mainメソッドからHumanクラスのメソッドeatを引数”パン”で呼ぶ

１.mainメソッドのあるクラスと別にHumanクラスを作成

```
public class Main{
        public static void main(String[] args){
        }
}


class Human{
}
```

１.mainメソッドのあるクラスと別にHumanクラスを作成

```
public class Main{
        public static void main(String[] args){
        }
}

class Human{
        String food;
        void eat(String food){
                this.food = food;
        }
}
```

Humanクラス

フィールド変数

・String型で変数名　food

メソッド

・eatメソッド

引数: String型1つ

処理: 変数foodに代入する

返り値なし

１.mainメソッドのあるクラスと別にHumanクラスを作成

```
public class Main{
        public static void main(String[] args){
        }
}

class Human{
        String food;
        void eat(String food1){
                food = food1;
        }
}
```

Humanクラス
フィールド変数
・String型で変数名　food

メソッド
・eatメソッド
引数: String型1つ
処理: 変数foodに代入する
返り値なし

2.mainメソッドからHumanクラスのメソッドeatを引数”パン”で呼ぶ

```
public class Main{
        public static void main(String[] args){
                Human human1 = new Human();
                human1.eat(“パン”);
        }
}

class Human{
        String food;
        void eat(String food1){
                food = food1;
        }
}
```

インスタンス生成

eat呼び出し

2. mainメソッドからHumanクラスのメソッドeatを引数”パン”で呼ぶ

```
public class Main{
	public static void main(String[] args){
		Human human1 = new Human();
		human1.eat(“パン”);
		System.out.println(human1.eat);
	}
}

class Human{
	String food;
	void eat(String food1){
		food = food1;
	}
}
```

インスタンス生成

eat呼び出し

```
public class Main{
	public static void main(String[] args){
		Human human1 = new Human();
		Human human2 = new Human();
		human1.eat(“パン”);
		human2.eat(“ご飯”);
		System.out.println(human1.eat);
		System.out.println(human2.eat);
	}
}
class Human{
	String food;
	void eat(String food1){
		food = food1;
	}
}
```

インスタンス生成

```
パン
ご飯
```

```
public class Main{
	public static void main(String[] args){
		Human human1 = new Human();
		human1.food = “パン”;
	}
}
class Human{
	String food;
	void eat(String food1){
		//もしfood1がパンならご飯に変える
		food = food1;
	}
}
```

インスタンス生成

Scanner : キーボード入力

Scanner sc　＝ new Scanner(System.in);

int a = sc.nextInt();

※クラスの外に

import△java.util.Scanner;

の記述が必要

```
import java.util.Scanner;

public class Main{
    public static void main(～){
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
    }
}
```

演習

1.mainメソッドのあるクラスと別にHelloクラスを作成

- Helloクラスの仕様

フィールド変数:　int型で変数名cnt
メソッド:
　　引数なし, 返り値なし
　　処理: 呼ばれたらcnt変数の値を+1する
　　“Hello　world” という文字列とcnt変数の値を出力する

2.mainメソッドからHelloクラスのインスタンス生成し、そのインスタンスでHelloクラスのメソッドを5回呼ぶ

演習２

1.mainメソッドのあるクラスと別にEnemyクラスを作成

- Enemyクラスの仕様

フィールド変数:　　int型で変数名hp
メソッドAttack：
　　引数　int型1つ, 返り値なし
　　処理: hpの値を引数で引く
　　　　hpが0以下なら “クリア” と出力
　　　　hpが0より多いなら “失敗” を出力

2.mainメソッドからEnemyクラスのインスタンス生成し、そのインスタンスで変数hpを適当な値に設定

3. mainメソッドでキーボードから値を入力して　２　のインスタンスからメソッドAttackを呼ぶ