Java講習 10

クラス　その３　コンストラクタ

復習

メソッド　・・・処理のまとまり

メソッド名(渡す値);

で呼べる

```
public class Main(){
  public static void main(String[] args){
            method1(1);      //メソッドを呼ぶ
  }
  void method1(int i){      //メソッド
            //処理
  }
}
```

メソッドのつくり方

```
返り値の型△メソッド名（受け取る値の変数宣言）{
        //処理
}
```

※返り値がないなら型の場所は void

復習　メソッド・・・処理のまとまり

メソッド名(渡す値);

で呼べる

```
public class Main(){
  public static void main(String[] args){
            method1(1);      //メソッドを呼ぶ
  }
  void method1(int i){      //メソッド
            //処理
  }
}
```

メソッドのつくり方

```
返り値の型△メソッド名（受け取る値の変数宣言）{
        //処理
}
```

※返り値がないなら型は void

クラス　・・・メソッド、変数の塊

使うには new クラス名(); をして、インスタンス生成する必要がある。

クラスはあくまで**設計図（テンプレート）**
中身が無い　→　詰める必要がある
　詰める作業(クラスを使う準備)　のことを**インスタンス生成**　という

ClassA　classA ＝ new ClassA();
インスタンス生成

コンストラクタ

コンストラクタ

new [クラス名]();

インスタンス生成時の初期状態を操れる！

インスタンス生成時、自動的に実行されるメソッド
フィールド変数などの初期化などに使う

記述としてはメソッドみたいに記述

インスタンス生成時に呼び出されるメソッド
→コンストラクタ

コンストラクタの書き方

```
クラス名(){
    //処理
}
```

※返り値は作れない

```
public class Main{
        public static void main(String[] args){
                Sub sub = new Sub();    //インスタンス生成し、
                                        //コンストラクタを呼ぶ
        }
}

class Sub{
        Sub(){//コンストラクタ
                System.out.println(“初期処理”);
        }
        void subMethod(){
        }
}
```

初期処理

```
クラス名(){
        //処理
}
```

コンストラクタ

通常のメソッドのように値を渡すこともできる

インスタンス生成　（値を渡す）

```
クラス名　変数　＝ new クラス名(渡す値);
```

コンストラクタ作成　（値を受け取る）

```
クラス名(貰う値の変数宣言){
　　//処理
}
```

コンストラクタ

通常のメソッドのように値を渡すこともできる

インスタンス生成　（値を渡す）　複数でも渡せる

```
クラス名　変数　＝ new クラス名(渡す値, 渡す値2);
```

コンストラクタ作成　（値を受け取る）

```
クラス名(貰う値の変数宣言1,貰う値の変数宣言1){
    //処理
}
```

```
public class Main{
        public static void main(String[] args){
                Sub sub = new Sub(1);   //インスタンス生成し、
                                        //コンストラクタを呼ぶ
        }
}

class Sub{
        Sub(int i){    //コンストラクタ
                System.out.println("初期処理" + i);
        }
        void subMethod(){
        }
}
```

初期処理1

```
クラス名(){
        //処理
}
```

どういう時に使うか

```
public class Main{
        public static void main(String args[]){
                Television tv1 = new Television();
                tv1.dispChannel();
        }
}
class Television{
        String name;
        void setName(String name2){
                name = name2;
        }
        void dispName(){
                System.out.println(“名前は" + name+ "です");
        }
}
```

```
public class Main{
        public static void main(String args[]){
                Television tv1 = new Television();
                tv1.dispChannel();
        }
}
class Television{
        String name;
        void setName(String name2){
                name = name2;
        }
        void dispName(){
                System.out.println("名前:" + name);
        }
}
```

名前: null

最初中身が何もない

```
public class Main{
        public static void main(String args[]){
                Television tv1 = new Television(“AAA”);
                tv1.dispChannel();
        }
}
class Television{
        String name;
        Television(String name2){
                neme = name2;
        }
        void setName(String name2){
                name = name2;
        }
        void dispName(){
                System.out.println(“名前は" + name+ "です");
        }
}
```

名前: AAA

例題:

(1) mainメソッドのあるクラスとは別にEnemyクラスを作成

Enemyクラスの仕様

フィールド変数　: int型　hp

コンストラクタ　: 引数の値をフィールド変数hpに代入

メソッド　　:フィールド変数hp を出力するメソッド

(2) mainメソッドからEnemyクラスのインスタンスを3つ生成

それぞれの引数を順にint型の 5, 10, 15　とする。

Enemyクラスの仕様

フィールド変数　　：int型　hp

コンストラクタ　　：引数の値をフィールド変数hpに代入

メソッド　　：フィールド変数hp を出力するメソッド

```
public class Main{
        public static void main(String[] args){
        }
}
class Enemy{
        int hp;                         //フィールド変数
        Enemy(int hp2){                 //コンストラクタ
                hp = hp2;
        }
        void printHp(){                 //メソッド
                System.out.println(hp);
        }
}
```

(2) mainメソッドからEnemyクラスのインスタンスを3つ生成

```
public class Main{
	public static void main(String[] args){
		Enemy enemy1 = new Enemy();		//インスタンス生成
		Enemy enemy2 = new Enemy();
		Enemy enemy3 = new Enemy();
	}
}
class Enemy{
	int hp;					//フィールド変数
	Enemy(int hp2){				//コンストラクタ
		hp = hp2;
	}
	void printHp(){				//メソッド
		System.out.println(hp);
	}
}
```

それぞれの引数を順にint型の 5, 10, 15　とする。

```
public class Main{
	public static void main(String[] args){
		Enemy enemy1 = new Enemy(5);		//インスタンス生成
		Enemy enemy2 = new Enemy(10);
		Enemy enemy3 = new Enemy(15);
	}
}
class Enemy{
	int hp;				//フィールド変数
	Enemy(int hp2){				//コンストラクタ
		hp = hp2;
	}
	void printHp(){			//メソッド
		System.out.println(hp);
	}
}
```

static について　[修飾子]

static をつけるとクラス固有のものになる。

→　new　せずに使える
　　いくらインスタンス生成しても1つしか存在しない
　　どこからでも共通にアクセスできる

staticメソッド　static変数

使い方

クラス名.メソッド名();

クラス名.変数名;

Keyboard.intValue();

```
public class Keyboard{
	static	int	intValue(){
		//int型の値をキーボードから受け取って返す
	}
	static	String	stringValue(){
		//int型の値をキーボードから受け取って返す
	}
}
```

```
int a = Keyboard.intValue();
```

クラスの継承

あるクラスをベースとして，それに機能を追加した別のクラスを作成することがある。

このとき、クラスの継承機能を用いることで簡単にその別クラスを作成することが出来る。　ヤッタネ！

```
class Car{
        int spped;
        void accele(){
                //処理
        }
        void brake(){
                //処理
        }
}
```

```
class Bus{
        int speed;
        void door(){
                //処理
        }
        void accele(){
                //処理
        }
        void brake(){
                //処理
        }

}
```

```
class Car{
        int speed;
        void accele(){
                //処理
        }
        void brake(){
                //処理
        }
}
```

土台を作っておく

```
class Bus{
        int speed;
        void door(){
                //処理
        }
        void accele(){
                //処理
        }
        void brake(){
                //処理
        }

}
```

継承方法

class△クラス名△**extends△土台のクラス名**{
}

土台

```
class Car{
        int speed;
        void accele(){
                //処理
        }
        void brake(){
                //処理
        }
}
```

```
class Bus extends Car{
        void door(){
                //処理
        }
}
```

土台

```
class Car{
        int speed;
        void accele(){
                //処理
        }
        void brake(){
                //処理
        }
}
```

```
class Bus extends Car{
        void door(){
                //処理
        }
}
```

```
class Bus {
        int speed;
        void door(){
                //処理
        }
        void accele(){
                //処理
        }
        void brake(){
                //処理
        }

}
```

参考サイト

JavaDrive　・・・Java全般(基礎的なことから～)
http://www.javadrive.jp/start/

Javaのオブジェクト指向入門　　・・・応用(主にクラス)
http://www.kab-studio.biz/Programing/OOPinJava/

演習１
(1)mainメソッドのあるクラスとは別にTvクラスを作成

Tvクラスの仕様
フィールド変数: String型 location
コンストラクタ: String型の引数
処理: 受け取った値をlocationに代入
locationの値を出力

(2) mainメソッドでTvクラスのインスタンスを2つ生成する。
引数は1つ目には”リビング”
２つ目には”和室”　とする。

演習2

Scanner：キーボード入力

コンストラクタの引数

```
Scanner　sc　＝ new　Scanner(System.in);
int a = sc.nextInt();
```

Scanner クラスのメソッド

※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();
    }
}
```

2

(1) mainメソッドのあるクラスと別にNumクラスを作成

- **Numクラスの仕様**

フィールド変数: int型 goal

コンストラクタ: 引数 int型,
処理: 引数をgoalに代入

メソッド: 引数int型, 返り値なし
処理: 引数とgoalを比べて、
引数の方が大きいなら”大きすぎます” と出力
引数の方が小さいなら”小さすぎます”と出力
同じなら”正解です” を出力

2

(2) mainメソッドからEnemyクラスのインスタンスを生成
　　この時、適当な値を引数として渡す

(3) mainメソッドで**キーボード入力**して、(2)のインスタンスから
　Enemyクラスのメソッドを呼ぶ
　　引数はキーボード入力した値

修飾子　例public, private,protected,etc･･･

クラス、メソッド、変数などには修飾子をつけることが出来る。
修飾子をつけることでアクセスの制限をすることなどができる。

＊public,privateなどの修飾子は特にアクセス修飾子と呼ばれる。
　それ以外の修飾子にはstatic,final,abstractなんかがある。

復習?

・アクセス修飾子
public…どっからでもアクセスできる。
protected…同じパッケージの中、継承してるサブクラス間限定。
private…同じクラスの中限定。
何もなし…同じパッケージの中だけ。

ほかの修飾子

特殊なものがいくつかあるので要注意

- final…最初に入れた値を変えることが出来ない。固定される。
  入れたが最後、そいつはもう助からない 。
- abstract…抽象化するやつ。　先輩に聞いてね
- static…メモリ上に残る頑固なやつ。静的変数。通常の変数は、インスタンスごとに異なる値を保持し、インスタンス変数と呼ばれる。一方、静的変数はインスタンスによらず共通のメモリ領域を占有する。クラスＡをインスタンス化したインスタンス１とインスタンス２があるときに、インスタンス１が静的変数を１０にセットしたら、インスタンス２から参照しても１０になっている。静的変数の利用にはインスタンス化の必要がない。
  http://msugai.fc2web.com/java/modify.html　参照