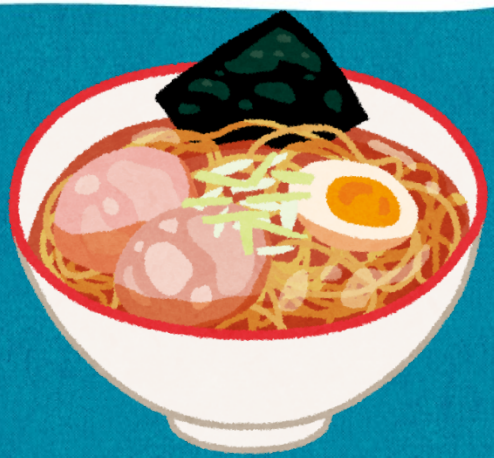
Java Silver 勉強会
第6章

継承とポリモフィズム

1.継承



継承とは

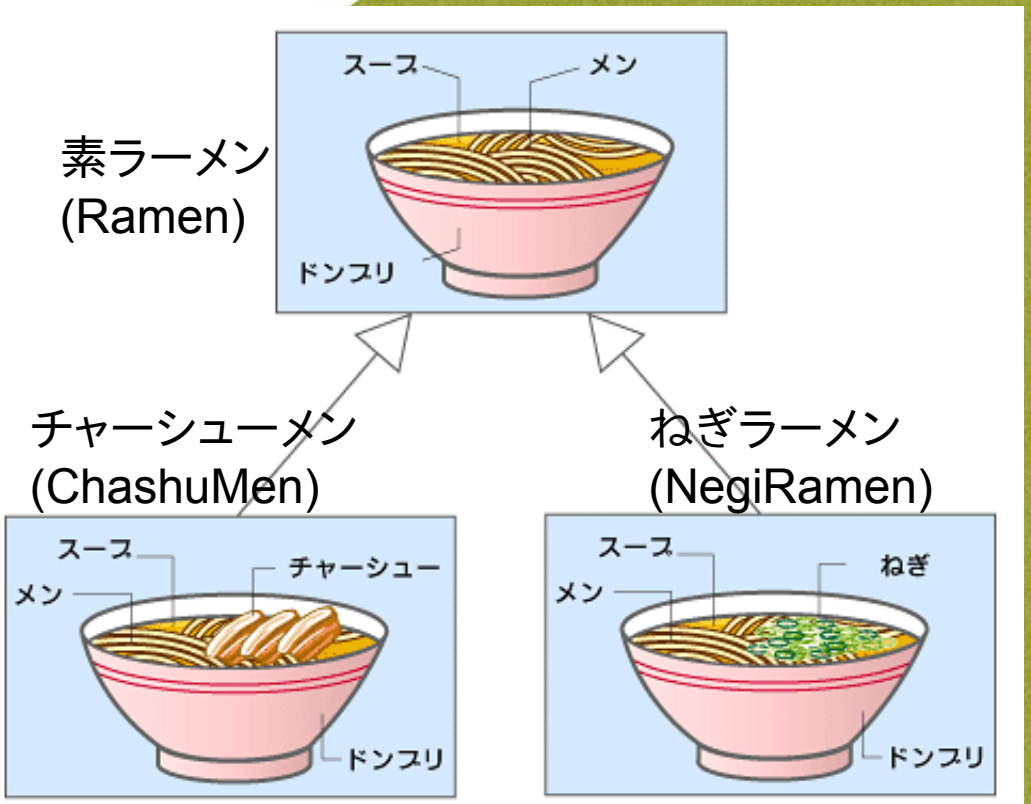
既存のクラス（スーパークラス）を元に
新たなクラス（サブクラス）を定義すること。

```
//スーパークラス
class Ramen { }

//サブクラス
① class ChashuMen extends Ramen{ }
② class NegiRamen extends Ramen{ }
```

継承とは

・スーパークラスで定義した変数、
　メソッドはすべてサブクラスに
　引き継がれる。
・サブクラス独自で変数やメソッドを定義。



素ラーメンは
麺とスープだけ

```
class Ramen {
    String name; //名前
    Men men;    //麺
    Soup soup; //スープ
    setName(…){…};
    setMen(…){…};
    setSoup(…){…};
}
```

麺＆スープ
＋チャーシュー

```
class ChashuMen extends Ram
    int chashu;   //チャーシュー
    setChashu (…){…};
}
```

麺＆スープ
＋ねぎ

```
class NegiRamen extends Ram
    int Negi;        //ねぎ
    setNegi (…){…};
}
```

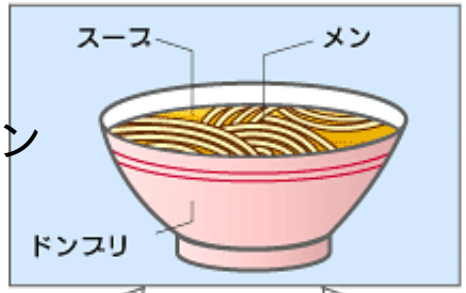
継承とは

素うどん

複数のクラスを継承するのはムリ

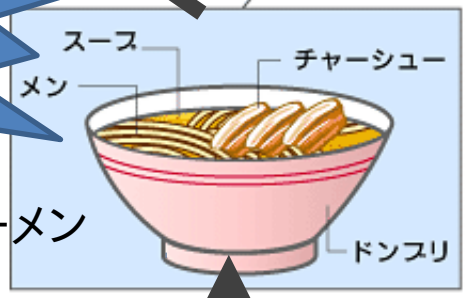
スーパークラス

素ラーメン

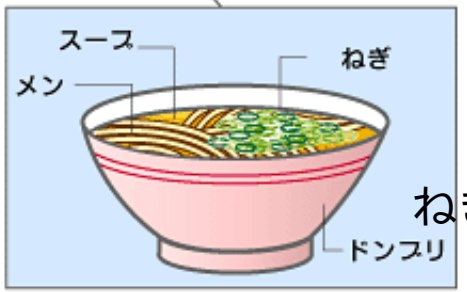


継承

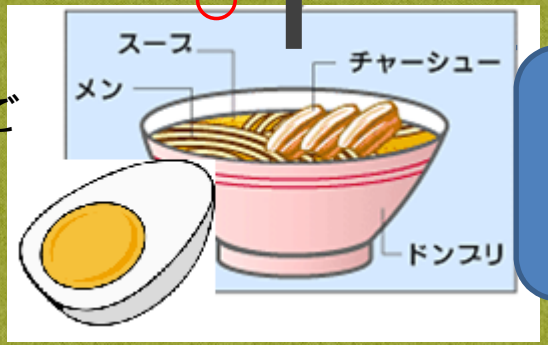
サブクラス



チャーシューメン



ねぎラーメン



チャーシュー＆煮たまごラーメン

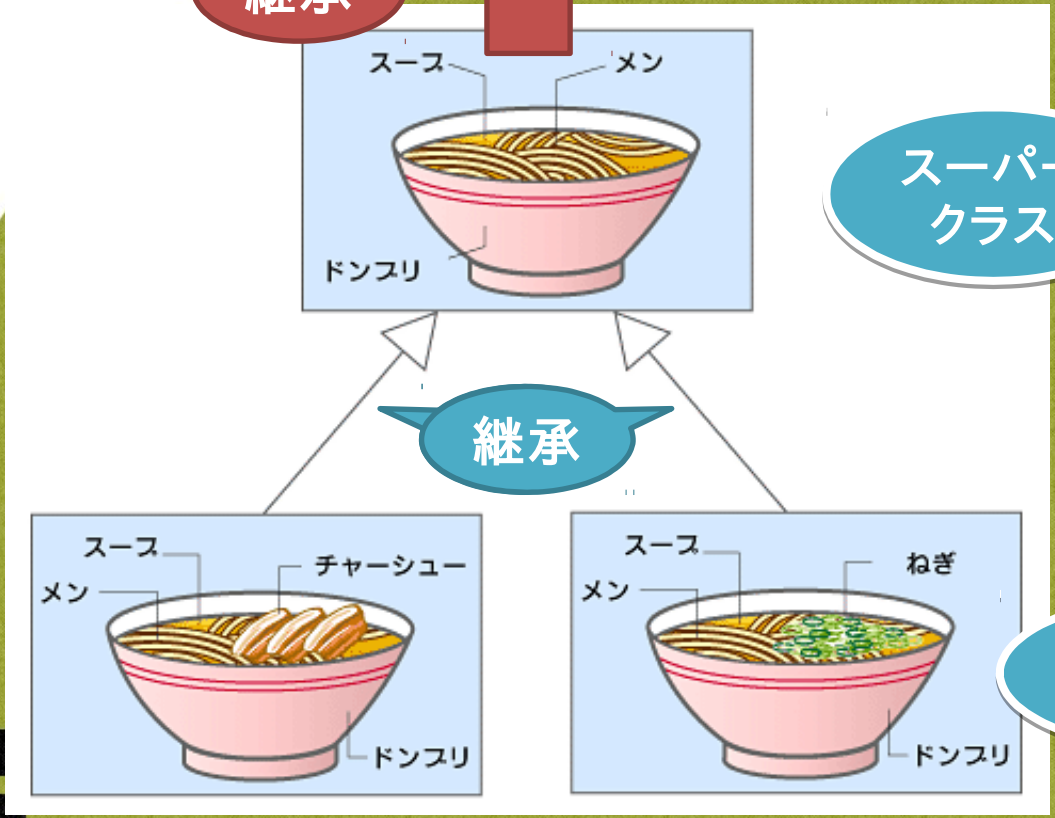
サブクラスから、さらにサブクラスを定義することはできる

継承とは

すべての
クラスの親

java.lang.Objectクラス

継承



extendsキーワードを
使用しないクラスは、
暗黙で
java.lang.Objectクラス
をスーパークラスにもつ。

オーバーライド

スーパークラスで定義しているメソッドと同じ名前のメ
サブクラス内に再定義すること。

素ラーメン:おいしい!

チャーシューメン:チャーシューおいしい!

```
class Ramen {
   public void print(String s)
      System.out.print("素ラーメン:"+s);
   }
}
class ChashuMen extends Ramen {          //サブクラス
   public void print(String s){
      System.out.print("チャーシューメン:チャーシュー"+ s ) ;
   }
}
class RamenShoukai{                      //実行クラス
   public static void main(String args[]){
      Ramen ramen = new Ramen();
      ramen.print("おいしい!");
      ChashuMen chashumen = new Chashumen();
      chashumen.print ("おいしい!");
   }
}
```

【メソッド名】
まったく同じ

【引数リスト】
まったく同じ

【アクセス修飾子】
同じか、より公開範囲が広いもの

【戻り値】
同じか、そのサブクラス型

final修飾子

- メソッドにfinal修飾子をつける
  [アクセス修飾子] final 戻り値の型 メソッド名(引数リスト) { }

  ⇒サブクラスでオーバーライドできなくなる

- クラスにfinal修飾子をつける
  [アクセス修飾子] final class クラス名 { }

  ⇒サブクラスが作れなくなる

thisとsuper
～this～

■this…自分自身（自オブジェクト）を指す

インスタンス変数
ローカル変数

null

```
① String name;
    void setName(String name){
        name = name;
    };
```

“白丸元味”

```
② String name;
    void setName(String name){
        this.name = name;
    };
```

“白丸元味”

thisとsuper
～this～

■this()→コンストラクタの中から、自クラスの別コンストラクタを呼び出す

呼ばれる側

```
class Order {
    String s;  int katasa;
    public Order() {
      this( “おまかせ” );
    }
    public Order(String s) {
      this( s, 1 );
    }
    public Order(String s , int i){
        this.s = s; int i = i;
        System.out.println(“商品:”
                              + ”    硬さ
    }
}
```

呼び出し側

```
class ChumonSuru {
   public static void main(String args[]){
     Order Asan = new Order();
     Order Bsan = new Order(“味噌ラーメン”)
    Order Csan = new Order(“醤油ラーメン”, 3
    }
}
```

実行結果

```
商品:おまかせ    硬さ:1

商品:味噌ラーメン    硬さ:1

商品:醤油ラーメン    硬さ:3
```

thisとsuper
～super～

■super…スーパークラスのオブジェクトを指す

```
① class Ramen {
        int price;
        public void methodA() {
           System.out.println(“ラーメン”);
        }
    }
② class Chashumen extends Ramen{
      public void methodA() {
           System.out.println(“チャーシュー”);
      }
       public void methodB() {
          methodA();
          super.methodA();
       }
    }
```

実行結果

チャーシュー
ラーメン

thisとsuper
～super～

■継承関係のあるクラスを普通にインスタンス化すると、スーパークラスの引数なしコンストラクタが実行されてからサブクラスのコンストラクタが実行される。

```
class Ramen {
    public Ramen() { System.out.println(“ラーメン”); }
    public Ramen( int a ) { System.out.println(“ラーメン（引数）”); }
}

class ChashuMen extends Ramen {
    public ChashuMen() { System.out.println(“チャーシューメン”); }
    public ChashuMen( int a ) { System.out.println(“チャーシューメン（引
数）”); }
}

class Aiueo{
    ChashuMen c1 = new ChashuMen
    ChashuMen c2 = new ChashuMen
}
```

実行結果

ラーメン
チャーシューメン

ラーメン
チャーシューメン（引数）

ラーメン（引数）は呼び出されない

thisとsuper
～super～

■super()→明示的にスーパークラスのコンストラクタを呼び出す
（⇒呼び出すコンストラクタを指定できる）

```
class Ramen {
    public Ramen() { System.out.println(“ラーメン”); }
    public Ramen( int a ) { System.out.println(“ラーメン（引数）”); }
}

class ChashuMen extends Ramen {
    public ChashuMen() { System.out.println(“チャーシューメン”); }
    public ChashuMen( int a ) {
       super(a);
        System.out.println(“チャーシューメン（引数
}

class Aiueo{
    ChashuMen c1 = new ChashuMen();
    ChashuMen c2 = new ChashuMen(10);
}
```

実行結果

ラーメン
チャーシューメン

ラーメン（引数）
チャーシューメン（引数）

thisとsuper

※this()もsuper()もコンストラクタ定義の先頭に記述する必要がある。

```
public
ChashuMen(int a){
    super(a);
    this();
}
```

どっちも先頭に書かないとコンパイルエラー
⇒両方同時に呼び出すのはムリ

```
public ChashuMen(){
    super(a);
}

public ChashuMen(int a){
    this();
}
```

２.抽象クラスとインタフェース

抽象クラス

■具象クラス
これまで出てきてたようなクラス。処理内容を記述したメソッド（具象メソッド）しか持たない。

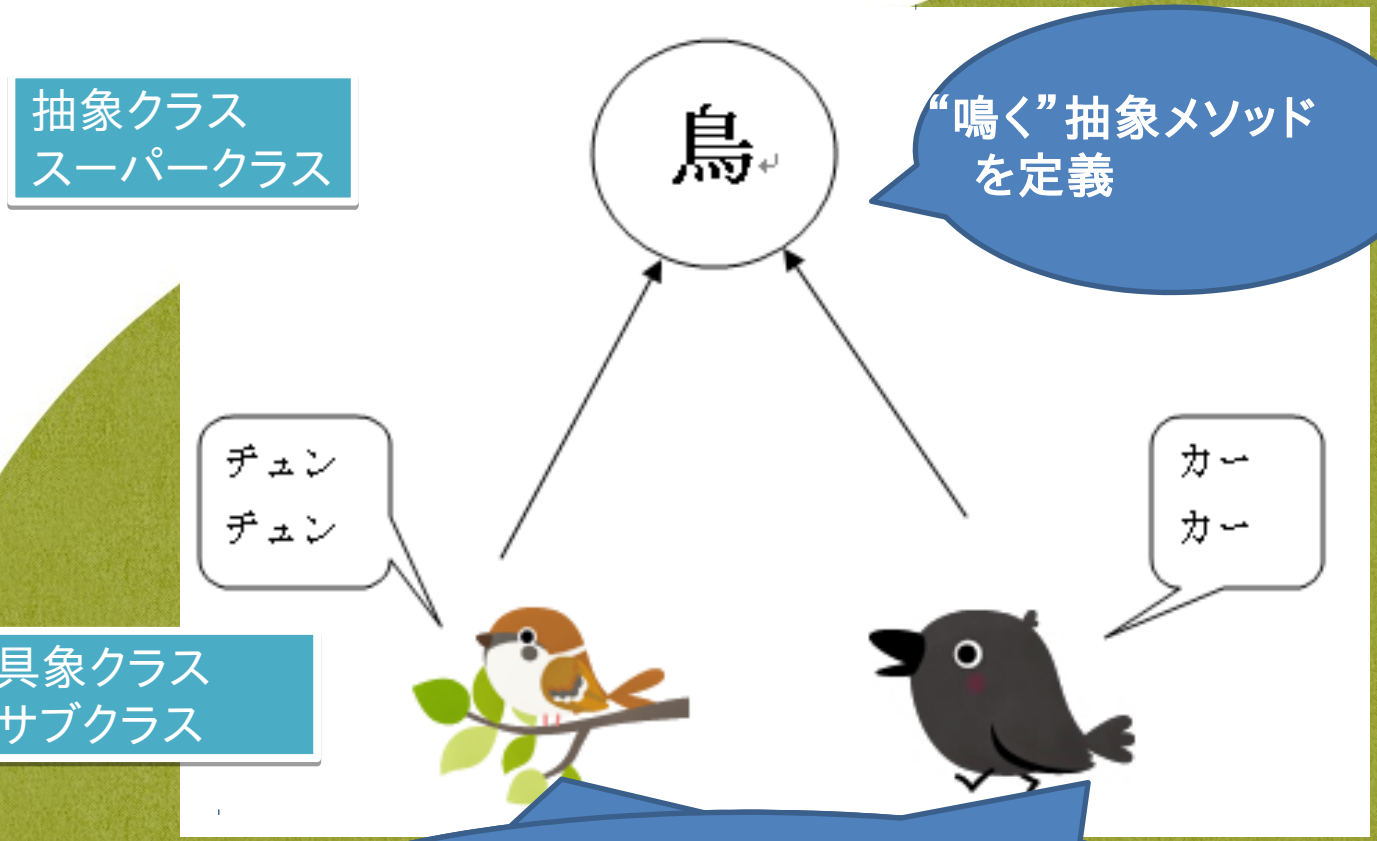
■抽象クラス

抽象クラス構文　[修飾子] abstract class クラス名 { }

- 具象メソッドも抽象メソッド（処理内容を記述しないメソッド）も持てる。
- インスタンス化できない。
  （抽象クラスを継承したサブクラスをインスタンス化して使う。）
- 抽象クラスを継承したサブクラスが具象クラスの場合、
  抽象クラスの抽象メソッドをすべてオーバーライドしないとだめ。
- 抽象クラスを継承したサブクラスが抽象クラスの場合、
  抽象クラスの抽象メソッドはオーバーライドしてもしなくてもいい。

抽象メソッド構文　[修飾子] abstract 戻り値 メソッド名(引数リスト);

抽象クラス



インタフェース

■インタフェース

構文　[修飾子] interface インタフェース名 { }

- インスタンス化できない。
- 抽象メソッドをオーバーライドする実装クラスを作成して使う。
- 実装クラスを定義するにはimprementsキーワードを使用
- インタフェースを元にサブインタフェースを作るときは
  extendsキーワードを使用
- インタフェースでは public static な定数を宣言できる。
（初期化が必須）
- SE8からは、抽象メソッドだけでなく、デフォルトメソッドと
staticメソッドも定義できるようになった。

インタフェース

インタフェースでの定数

⇒ staticな変数のみ。
　インスタンス変数は宣言できない。
必要な修飾子を記述しなくても、強制的に
public static final 修飾子が付与される。
初期化しておかないとコンパイルエラー。

インタフェース

インタフェースでのメソッド

①抽象メソッド ⇒ 必要な修飾子を記述しなくても、強制的に public abstract 修飾子が付与される。

②デフォルトメソッド 具象メソッドが定義できる！

⇒ 構文　default 戻り値 メソッド名(引数リスト) { 処理 }

※指定できる修飾子はpublicのみ。

（指定しないと強制的にpublic修飾子が付与される）

③staticメソッド

（5章参照）

※指定できる修飾子はpublicのみ。

（指定しないと強制的にpublic修飾子が付与される）

インタフェース

インタフェースの実装クラス

インタフェースで宣言された抽象メソッドをオーバーライドするクラス

構文　[修飾子] class クラス名 implements インタフェース名 { }

複数指定することもできる。

※実装クラスではimplementsキーワードで指定したすべての
　インタフェースの抽象メソッドをオーバーライドする
※オーバーライドするときは public修飾子が必要
　（インタフェースの抽象メソッドは暗黙でpublic abstractだから）
※実装クラスはインタフェースの実装（implements）と
　他クラスの継承（extends）を同時に行うこともできる。
　（extendsを先に書かないとだめ）

インタフェース

インタフェースの継承

⇒ あるインタフェースを元にして、サブインタフェースを宣言できる。

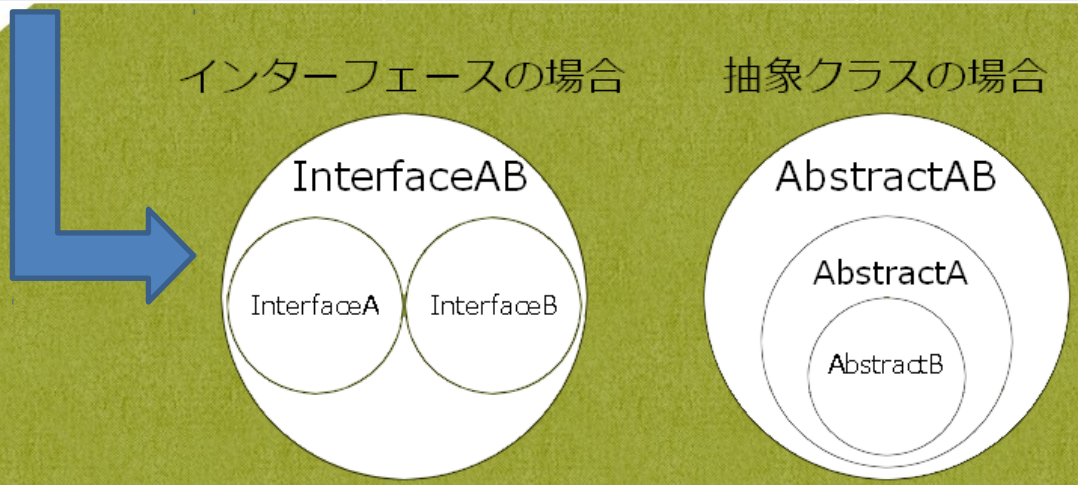
extendsキーワードを使用

※サブインタフェースを実装した具象クラスは、
スーパーインタフェース・サブインタフェースの抽象メソッドを
オーバーライドする必要がある

**※インタフェースは複数のインタフェースを
継承（extends）することができる**

抽象クラスとインタフェースのちがい

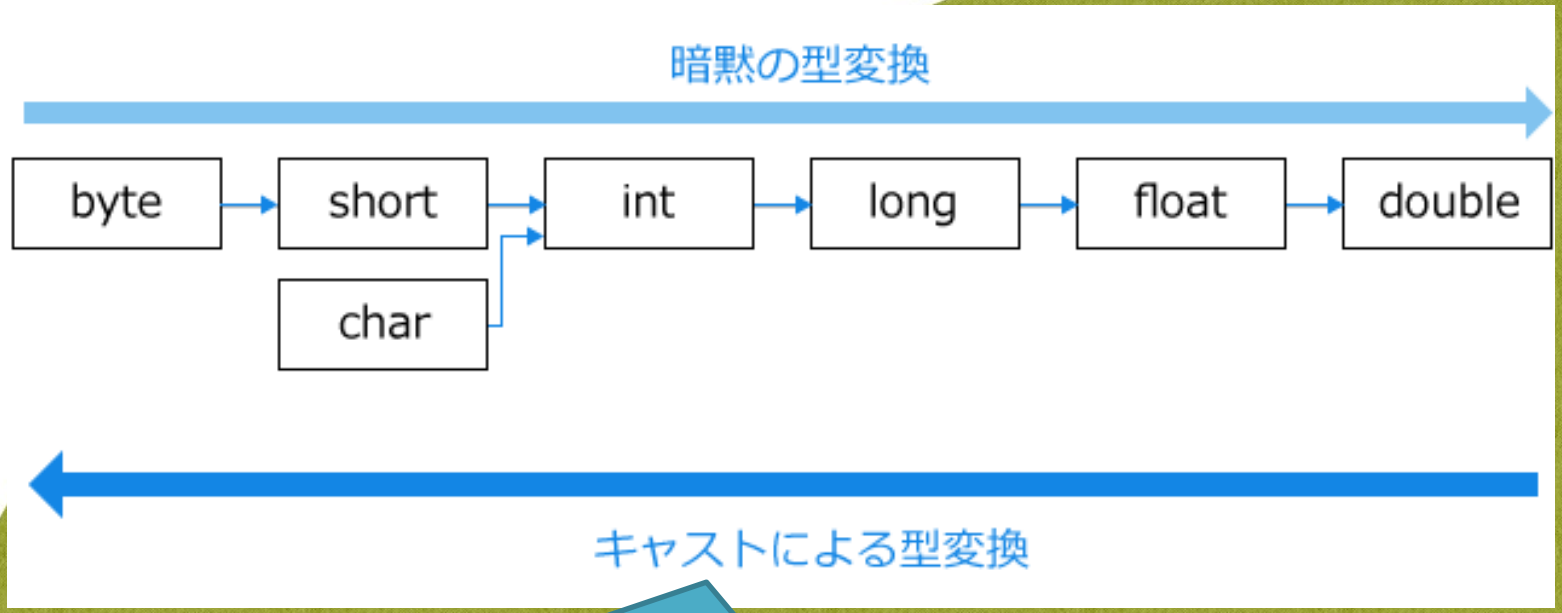
	抽象クラス	インタフェース
具象メソッド	持てる	SE8から持てるようになった
多重継承	できない	できる



抽象クラスとサブクラスの「継承」関係は「IS A」の関係で、
インタフェースと実装クラスの「実装」関係は「CAN DO」????

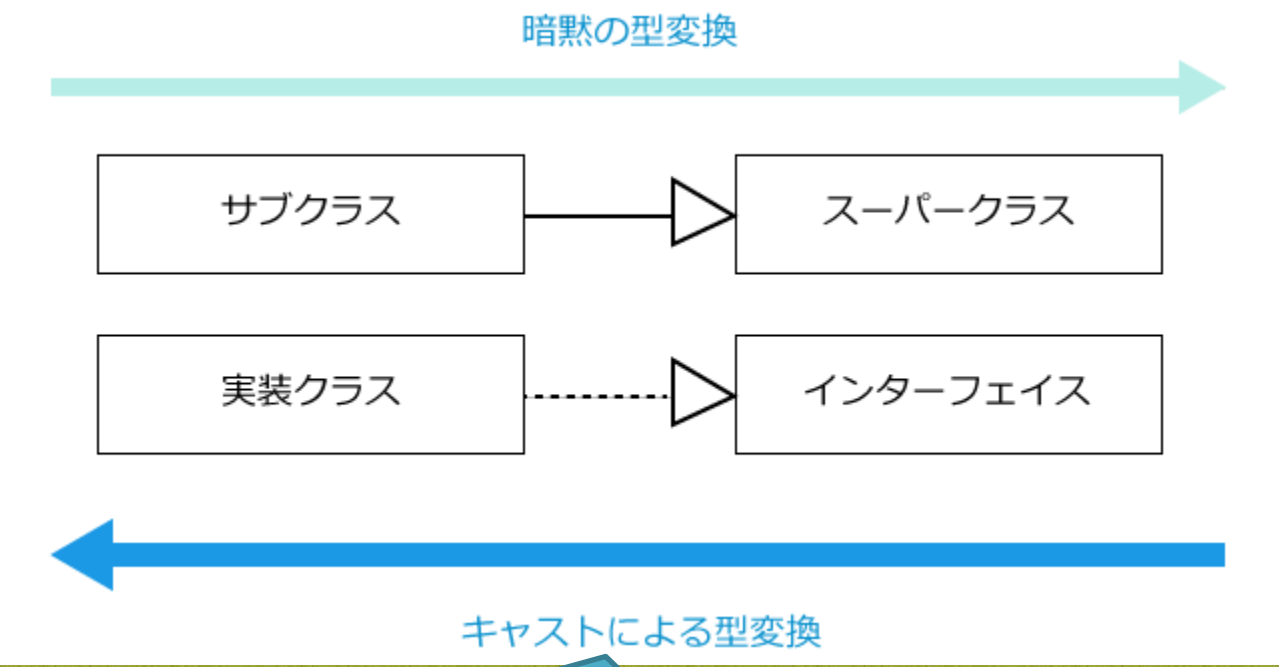
3. 型変換

基本データ型の型変換



【構文】　(目的の型) 値

参照型の型変換



【構文】　(目的の型) オブジェクト

参照データがキャストしようとしているクラスから作成されたオブジェクトでなければ、実行時にClassCastExceptionが発生する。

参照型の型変換

スーパークラス

```
class Super {
    void methodA() { }
}
```

サブクラス

```
class Sub extends Super  {
    void methodA() { }
    void methodB() { }
}
```

サブクラスを利用したクラス･例1

```
class Test {
    …
   Super s = new Sub();
    s.methodA();
    s.methodB();    //コンパイルエラー
    …
```

コンパイル時
SuperクラスにmethodA,Bがあるか
（⇒Bない。コンパイルエラー）

サブクラスを利用したクラス･例2

```
class Test {
    …
   Super s1 = new Sub();
    s1.methodA();
   Super s2 = new Sub();
    s2.methodB();
    …
}
```

実行時
SubクラスのmethodA,Bを呼び出し

instanceof演算子

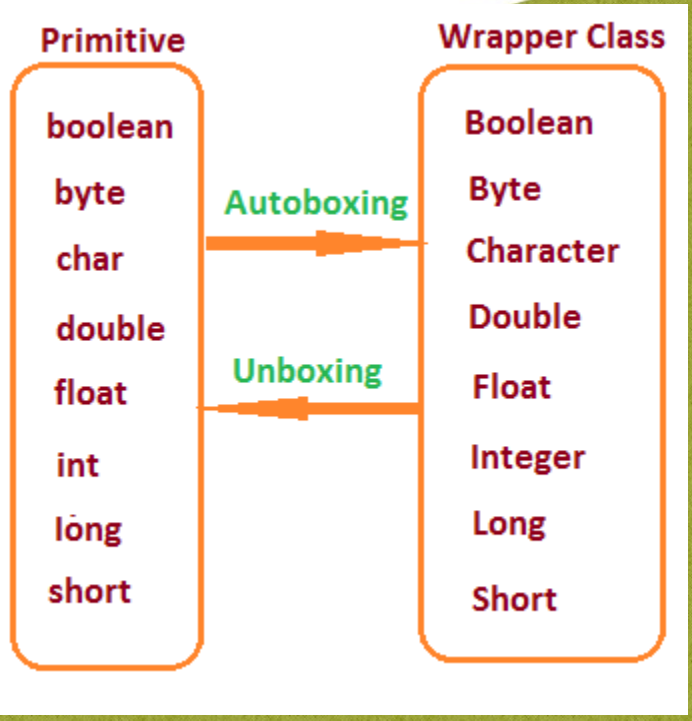
構文　参照変数名 instanceof クラス名またはインタフェース名

ある特定のオブジェクトが特定の値を持つかどうかを判定し、
結果をboolean値で返す。

左辺で指定した変数に、右辺で指定した型を持っていればtrue

※右辺・左辺に指定されたクラスには継承関係がなければいけない
※右辺にインタフェースを指定するときは、左辺はその実装クラスのオブジェクトじゃなくてもいい

オーバーロード時での注意



Autoboxingでは暗黙の型変換はできない。
（コンパイルエラーになる）

オーバーロードされたメソッドを呼び出す
優先順位は

完全一致＞暗黙の型変換
＞オートボクシング＞可変長引数