■基本事項

<> で囲まれた型は、ジェネリックス型、ジェネリックス型が適用された型をベース型。

型	多態性
ベース型	○
ジェネリックス型	×
配列	○

・配列自体に、多態性を適用した場合で、実行時に型が不一致の場合、ArrayStoreException となる。(1)
・ジェネリックスの場合、実行時に型情報が失われるため、ジェネリックスを利用したベース型の多態性の利用はコンパイルされない。(2)
・ジェネリックスを利用したベース型の要素格納等には当然多態性を利用できる。(3)

```
public class GenericsTest1 {
  public static void main(String[] args) {
    GenericsTest1 me = new GenericsTest1();
    // List, ArrayList はベース型、Deriv はジェネリックス型
    // ジェネリックス型に、多態性は適用されない
    // ArrayList<Number> l = new ArrayList<Integer>(); //NG (2)
    List<Integer> l2 = new ArrayList<Integer>(); //OK

    // 当然ながら、保持する要素レベルでは多態性は利用できる (3)
    List<Number> l3 = new ArrayList<Number>();
    l3.add(new Integer(1));
    l3.add(new Long(2));

    // 配列では、以下のように、配列の型自体に、多態性を使用できる (1)
    Number[] a = new Integer[2];

    a[0] = new Number() {
      private static final long serialVersionUID = 1L;
      public int intValue()       { return 0; }
      public long longValue()     { return 0; }
      public float floatValue()   { return 0; }
      public double doubleValue() { return 0; }
    };
    a[1] = new Integer(0); // ただし、現実の型と不一致の場合、ArrayStoreException が投げられる
  }
}
```

■ワイルドカード型 ＋ extend

・多態性を利用して、統一的に処理を行いたい (基底クラスが提供するメソッドを呼び出したい等) 場合は、ワイルドカード＋ extend を利用する。(1)
・引数型を、List<? extend Clazz> とすることで、 IS-A Cazz にパスする任意のジェネリックス型のリストを渡すことができる。
・ただし、このリストに要素を追加することはできない。(実際の型が何であるか不明)(2)
・削除操作などは可能。(3)

```
public class GenericsTest2 {
    public static void main(String[] args) {
        GenericsTest2 me = new GenericsTest2();
        List<Integer> l1 = new ArrayList<Integer>();
        List<Long> l2 = new ArrayList<Long>();

        l1.add(new Integer(1));
        l1.add(new Integer(2));
        l1.add(new Integer(3));

        l2.add(new Long(100));
        l2.add(new Long(200));
        l2.add(new Long(300));

        // Deriv1、Deriv2 何れのジェネリックス型のリストも渡すことができる
        me.printNames(l1);
```

```
        me.printNames(l2);
    }

    public void printNames(List<? extends Number> lst) {  // ワイルドカードの使用 (1)
        // lst.add(this.new Base("B-1"));  // リストに追加する操作は、コンパイルエラー (2)
        lst.remove(1);                     // 削除する操作は、OK (3)

        for(Number b : lst) {
            System.out.println(b);
        }
    }
}
```

■ワイルドカード型 ＋ super

- ? + extends とは逆に、多態性を利用して、基底クラスの参照に、派生クラスを格納したいような場合は、ワイルドカード＋ super を利用する。(1)
- 引数型を、List<? super Clazz> とすることで、Cazz IS-A ? にパスする任意のジェネリックス型のリストを渡すことができる。(2)
- 渡された側では、ジェネリックス型に代入互換性があるクラスを追加することができる。

```
import java.util.ArrayList;
import java.util.List;

public class GenericsTest3 {
    public static void main(String[] args) {
        GenericsTest3 me = new GenericsTest3();
        List<Base>       baselst     = new ArrayList<Base>();
        List<Deriv>      drvlst      = new ArrayList<Deriv>();
        List<Deriv_1>    drvlst_1    = new ArrayList<Deriv_1>();
        List<Deriv_2>    drvlst_2    = new ArrayList<Deriv_2>();
        List<Deriv_1_1>  drvlst_1_1  = new ArrayList<Deriv_1_1>();

        me.putObject(baselst);     // OK Base      に Deriv_1 は代入可能 (2)
        me.putObject(drvlst);      // OK Deriv     に Deriv_1 は代入可能 (2)
        me.putObject(drvlst_1);    // OK (2)
     // me.putObject(drvlst_2);    // NG Deriv_2   に Deriv_1 は代入互換性なし
     // me.putObject(drvlst_1_1);  // NG Deriv_1_1 に Deriv_1 は代入互換性なし
    }

    public void putObject(List<? super Deriv_1> lst) { // (1)
        Base base         = this.new Base();
        Deriv drv         = this.new Deriv();
        Deriv_1 drv_1     = this.new Deriv_1();
        Deriv_2 drv_2     = this.new Deriv_2();
        Deriv_1_1 drv_1_1 = this.new Deriv_1_1();

     // lst.add(base);     //NG Deriv_1 に Base は代入できない
     // lst.add(drv);      //NG Deriv_1 に Deriv は代入できない
        lst.add(drv_1);    //OK
     // lst.add(drv_2);         //NG Deriv_1 に Deriv_2 は代入できない
        lst.add(drv_1_1);  //OK Deriv_1 に Deriv_1_1 は代入できる
    }
    private class Base {}
    private class Deriv     extends Base    {}
    private class Deriv_1   extends Deriv   {}
    private class Deriv_1_1 extends Deriv_1 {}
    private class Deriv_2   extends Deriv   {}
}
```

■インスタンスを生成する

```
public <T> T createInstance(Class<T> cls) {
    return cls.newInstance();
}
```