http://www.ne.jp/asahi/hishidama/home/tech/java/generics.html
http://d.hatena.ne.jp/Nagise/20101105/1288938415

文法

例

■クラス

```
class GenericsTest <T, S, R> { ... }
```

■メソッド

```
public <T,S,R> T test(S s, R r) { ... return T}
// 呼び出し方
String s = クラス名 .<String, String, String>test("hoge", "hoge")
```

用語

■型変数

型を実行時（コンパイル時）に確定するもの。
上記の例の場合、 T や S、R をそれぞれ型変数という。

■型パラメータ

型変数のを宣言しているリスト。
上記の例の場合、<T,S,R> を型パラメータリストという。

文法上の注意

文法上混乱しやすいので注意。

使用箇所	宣言	バインド	?	境界	&
型変数の宣言 class test<T> ...	○	×	△	○	○
変数の型の宣言 Test<String> test	×	○	○	○	×
型変数へのバインド new Test<String>();	×	○	×	×	×

使い方

Generics の利点は

・キャストの省略
・型の保証

である。内部的には Object 型（または、指定している境界の型）として処理される。

キャストは自動的に行われ、キャストの際に指定した型として扱えるかチェックされる。

コンパイル後は型の情報は失われている。これを型のイレイジャと呼ぶ。

型の代入互換性

Generics では Java の型の代入互換性ルールが通常と異なる。

```
public class A {}
public class B extends A {}
public class C extends B {}

A a = new B();
B b = new C();
```

上記は、可能である。が以下はエラーになる。

```
Hoge<A> a = new Hoge<B>(); // コンパイルエラー！
```

理由は、もしこれが可能になると以下の場合に問題が出る。

```
ArrayList<B> bList = new ArrayList<B>();
ArrayList<A> aList = bList; // 本来は代入できないができたと仮定する
aList.add(new A()); // ArrayList<A> には A 型を代入できる
B b = bList.get(0); //ArrayList<B> なので get() の結果は B 型のはず
```

A クラスから B クラスへのアップキャストが発生してしまう。

これに対応するためにワイルドカードでの境界指定を行う。

```
Hoge<? extends A> a = new Hoge<B>();
```

<? extends A>

この場合、A クラスのサブクラスである何かであることを保証するが、型が不確定である。
そのため、

- 戻り値に型変数が使われている場合、A 型の返り値を保証する
- 引数に型変数があるメソッドを呼び出すことができなくなる (引数の型が不明であるため）

<? super A>

この場合、A クラスのスーパークラスである何かであることを保証するが、型が不確定である。
そのため、

- 戻り値に型変数が使われている場合、Object 型の返り値を保証する
- 引数に型変数があるメソッドに A 型（または、A 型のサブクラス）だけを渡すことを許可する