

Generics

Java Basic



Based On Java 21

Contents



- About Generics
- Type Parameters
- GenericTypes
- Generic Methods
- Wild Cards for Using GenericTypes
- Type Erasure

About Generics

- Generics ဟာ Programming Paradigm တစ်ခုဖြစ်ပြီး Java Language ထဲကို Java Version 5 အရောက်မှာ ပါဝင်လာခဲ့ပါတယ်
- Programming Module တစ်ခုဟာ အလုပ်လုပ်တာတွေတူညီပြီး အသုံးပြုမည့် Data Type တွေမတူခဲ့ရင် Type ကို Abstraction ဖြစ်အောင် ရေးသားလိုက်ခြင်းအားဖြင့် အလုပ်လုပ်အောင် ရေးသားထားတဲ့ Algorithm တွေကို Reuse လုပ်နိုင်လိမ့်မယ်ဆိုတာကတော့ Generics ရဲ့ အခြေခံ Concept ဖြစ်ပါတယ်
- Generics ကြောင့် Java မှာ Type တွေကို Abstraction ဖြစ်အောင် ရေးသားလာနိုင်ပြီး Type Check System ကိုလဲ ပိုမိုကောင်းမွန်အောင် ပြင်ဆင်လာနိုင်ခဲ့ကြပါတယ်

Before Generics

- Java Language မှာ Generics ကို မထည့်သွင်းခင်တုန်းက Type တွေအားလုံးကို အသုံးပြုတဲ့ Method တွေကို ရေးသားချင်ရင် Class တွေအားလုံးရဲ့ Super Class ဖြစ်တဲ့ Object ကို အသုံးပြုပြီး ရေးသားခဲ့ကြရပါတယ်
- Set လုပ်တဲ့ Method တွေမှာဆိုရင် Object ကို Argument အနေနဲ့ ယူထားရင် ဘယ်လို Type မျိုးမဆို အသုံးပြုလို့ရတဲ့ အတွက် အဆင်ပြေသလို ဖြစ်နေပါတယ်
- ဒါပေမဲ့ ပြန်ပြီးထုတ်ယူတဲ့ အခါမှာ Return Type ဟာလဲ Object ဖြစ်နေပြန်ပြီး နောက်ကွယ်က actual object ရဲ့ Type က ဘာမဆိုဖြစ်နေနိုင်ပြန်တယ်
- မိမိအသုံးပြုလိုတဲ့ Type အဖြစ် Type Casting လုပ်တဲ့ အခါကျမှ အခန့်မသင့်ရင် java.lang.ClassCastException ကို ဖြစ်စေခဲ့ပါတယ်

Class using Object



```
public class Wrapper {  
  
    private Object data;  
  
    public void setData(Object data) {  
        this.data = data;  
    }  
  
    public Object getData() {  
        return this.data;  
    }  
}
```

Problem of using object



```
Wrapper wrapper = new Wrapper();

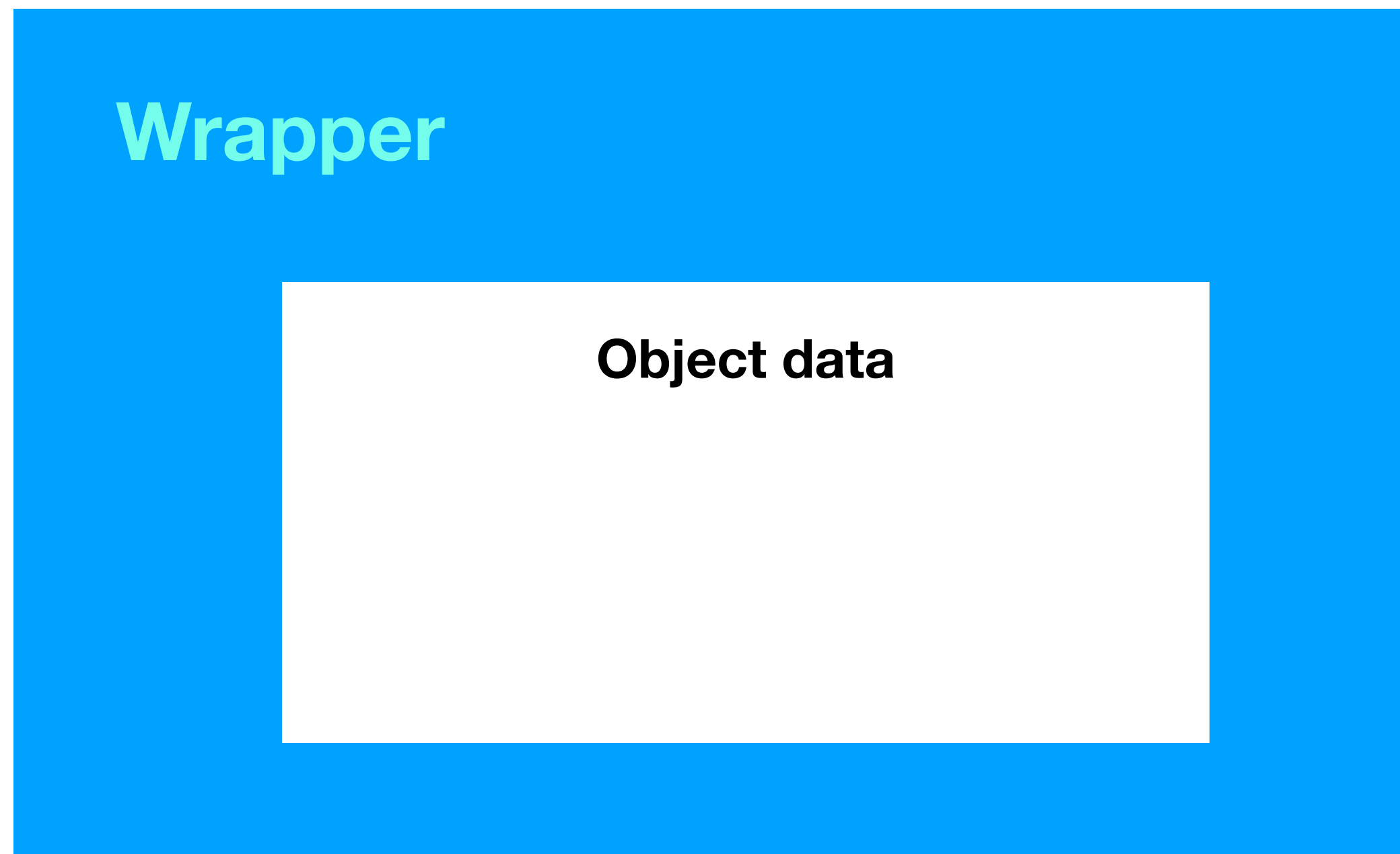
// Every Data Can set because of Object
wrapper.setData(new Date());
wrapper.setData("Hello World");

// But we can only get as Object
Object result = wrapper.getData();

// If we cast correctly it is OK
String str = (String)result;

// We get ClassCastException when we miss
Date date = (Date)result;
```

Where are the ugly things



By Using Generics

- Generics ကို အသုံးပြုရင် Type တွေကို Parameter အနေနှင့် သတ်မှတ်နိုင်ပြီး Object ဆောက်တဲ့ အခါကျမှ Type Parameter နေရာမှာ အသုံးပြုမည့် Type ကို သတ်မှတ်ပေးရပါမယ်
- အဲဒီအခါကျမှ Type Parameter များကို ရေးသားထားတဲ့ နေရာတိုင်းဟာ အသုံးပြုမည့် Type အဖြစ် ပြောင်းလဲ သွားမှာ ဖြစ်ပါတယ်
- ဤနည်းအားဖြင့် Setter Method ရဲ့ Argument နေရာမှာလဲ အသုံးပြုမည့် Type ဖြစ်သွားတဲ့ အတွက် အခြားသော Type များကို အသုံးပြုလို့ရတော့မှာမဟုတ်သလို၊ Getter Method ရဲ့ Return Type ဟာ လည်း အသုံးပြုမည့် Type ဖြစ်တဲ့အတွက် Type Cast လုပ်စရာလိုအပ်တော့မှာ မဟုတ်တော့ပါဘူး

Class using Generics



```
public class Wrapper<T> {  
    private T data;  
  
    public void setData(T data) {  
        this.data = data;  
    }  
  
    public T getData() {  
        return this.data;  
    }  
}
```

Because of Generics



```
// All plcase that's using type parameter will be Date  
Wrapper<Date> wrapper = new Wrapper<>();
```

```
// It will be OK as argument type is Date  
wrapper.setData(new Date());
```

```
// It can't allow  
wrapper.setData("Hello World");
```

```
// Return Type will be date, so no need to type casting  
Date result = wrapper.getData();
```

Generics Mechanism

Wrapper<T> class

```
private T data;
```

```
public void setData(T data)  
public T getData()
```

Create Object

Wrapper<Date> object

```
private Date data;
```

```
public void setData(Date data)  
public Date getData()
```

- GenericsType တွေကို Object ဆောက်တဲ့ အခါမှာ အသုံးပြုမည့် Type ကို သတ်မှတ်ပေးရပြီး သတ်မှတ်ထားတဲ့ Type ဖြင့် Type Parameter တွေရဲ့ နေရာကို အစားထိုးအသုံးပြုသွားမှာ ဖြစ်ပါတယ်

Type Parameters

- Generics တွေကို Class, Interface, Record နှင့် Method တွေမှာ ရေးသား အသုံးပြုနိုင်ပါတယ်
- Generics ကို အသုံးပြုမည့် Type တွေ Method တွေမှာ Type Parameter တွေကို ထောင့်ကွင်းလေး <TypeParameter> ဖြင့် ရေးသား နိုင်ပါတယ်
- Type Parameter Name အနေနဲ့ကတော့ သတ်မှတ်ထားတာမရှိပါဘူး၊ နှစ်သက်ရာ String ကို အသုံးပြုနိုင်ပါတယ် (Reserver Words နှင့် Keyword တွေကတော့ သုံးလို့မရပါဘူး)
- လိုအပ်ပါက Type Parameter တွေကို Comma ခံပြီး တစ်ခုထက်မက ရေးသား အသုံးပြုနိုင်ပါတယ်

Multiple Type Parameters



```
public class Pair<K, V> {  
  
    private K key;  
    private V value;  
  
    public Pair(K key, V value) {  
        this.key = key;  
        this.value = value;  
    }  
  
    public K getKey() {  
        return key;  
    }  
  
    public V getValue() {  
        return value;  
    }  
}
```

Bounded Type

- Generics ကို အသုံးပြုဖို့ Type Parameter တွေကို သတ်မှတ်ရေးသားတဲ့ အခါမှာ အသုံးပြုနိုင်တဲ့ Type တွေကို ကန့်သတ်လိုတဲ့ အခါတွေ အတွက် Type တွေကို Bounded လုပ်ထားနိုင်ပါတယ်
- Type တွေကို Bounded လုပ်လိုတဲ့ အခါမှာ Type Parameter ရဲ့ အနောက်မှာ `<T extends BoundedType>` ဆိုပြီး ရေးသားပေးကြရမှာ ဖြစ်ပါတယ်
- Bounded Type ကို အသုံးပြုပြီး ရေးသားထားပါက Bounded Type နှင့် Sub Type များသာ Type Parameter နေရာမှာ အစားထိုးအသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်
- ထို့အပြင် Type Parameter များကို Bound လုပ်ထားတဲ့ အတွက် Type Parameter များမှာ Bounded Type မှာပါတဲ့ Properties များကို အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်

Using Bounded Type



```
public class Wrapper<T extends Number> {  
  
    private T data;  
  
    public Wrapper(T data) {  
        super();  
        this.data = data;  
    }  
  
    public T getData() {  
        return data;  
    }  
  
    public int getIntValue() {  
        return data.intValue();  
    }  
}
```

Generic Types

- Type တွေအနေနှင့် အသုံးပြုနိုင်တဲ့ Class, Interface နှင့် Record တွေမှာ Generics ကို အသုံးပြုထားပါက Generic Type လို့ ခေါ်ဆိုလေ့ရှိပါတယ်
- Record တွေဟာ Default Final Entity တွေဖြစ်ကြတဲ့အတွက် Sub Type တွေကို လက်မခံနိုင်ပဲ၊ Class တွေနှင့် Interface တွေမှာတော့ Inheritance ကို လက်ခံနိုင်ပြီး Sub Type တွေကို ရေးသားအသုံးပြုနိုင်ကြပါတယ်
- Generic Type တွေရဲ့ Sub Type တွေကို ရေးသားတဲ့ နေရာမှာ Generic Type အနေနဲ့ပဲ ထားပြီး အသုံးပြုမလား ဒါမှမဟုတ် Type တွေကို သတ်မှတ်ပြီး အသုံးပြုမလား ဆိုပြီး ရွေးချယ်ကြရမှာ ဖြစ်ပါတယ်

Generic Interface



```
public interface Wrapper<T> {  
    void setData(T data);  
    T getData();  
}
```

Sub Type as a Generic Type



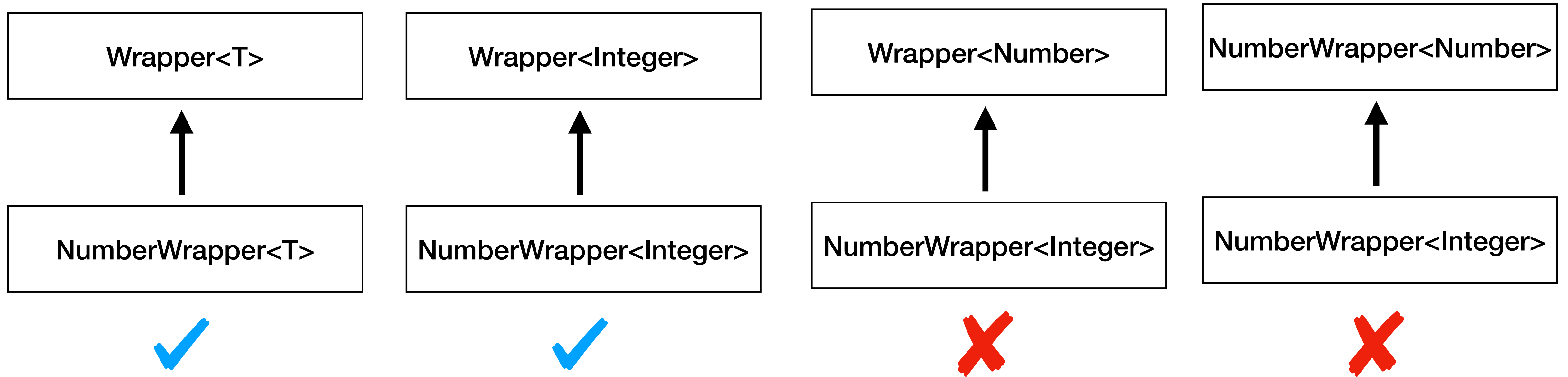
```
public class NumberWrapper<T> extends Number> implements Wrapper<T>{  
  
    private T data;  
  
    @Override  
    public void setData(T data) {  
        this.data = data;  
    }  
  
    @Override  
    public T getData() {  
        return data;  
    }  
  
}
```

Sub Type as Normal Class



```
public class DateWrapper implements Wrapper<Date>{  
  
    private Date data;  
  
    @Override  
    public void setData(Date data) {  
        this.data = data;  
    }  
  
    @Override  
    public Date getData() {  
        return data;  
    }  
}
```

Generic Types & Inheritance



- Generic Type တွေအကြားမှာ IS - A Relationship ရှိကြပေမဲ့ Type Parameter တွေမှာတော့ IS - A Relationship မရှိကြပါဘူး

Generic Methods

- Class, Interface နှင့် Record တွေမှာ Generics ကို အသုံးပြုလို့ရနိုင်သလို Method တွေမှာလဲ Generics ကို အသုံးပြုနိုင်ပါတယ်
- Generics တွေကို အသုံးပြုထားတဲ့ Method တွေကို Generic Method လို့ခေါ်ဆိုလေ့ရှိပါတယ်
- Executable တွေဖြစ်ကြတဲ့ Static Method တွေ Instance Method တွေနှင့် Constructor တွေ မှာပါ Generics ကို အသုံးပြုနိုင်ပြီး၊ Method တွေမှာ ရေးသားထားသော Type Parameter များကို Method Scope အတွင်းမှာသာ အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်
- Type Parameter တွေကိုတော့ Modifier နှင့် Return Type အကြားမှာ ရေးသားကြရမှာ ဖြစ်ပါတယ်

Writing Generic Method



```
public class FormattedStringUtils {  
    private static final DecimalFormat DF = new DecimalFormat("#,##0.00");  
  
    public static <T extends Number> String format(T data) {  
        var value = data.doubleValue();  
        return DF.format(value);  
    }  
}
```

Wild Card


- Generic Type တွေကို အသုံးပြုတဲ့အခါမှာ မည်သည့် Type ကို အသုံးပြုမယ်ဆိုတာကို မသတ်မှတ်ရသေးရင် Unknown Type ဖြစ်တဲ့ Wildcard (?) ကို အသုံးပြုပြီး ရေးသားနိုင်ပါတယ်
- Wild Card ကို Generic Type ကို အသုံးပြုထားတဲ့ Type Parameter တွေနေရာမှာ Method Arguments တွေ၊ Instance Variables တွေ၊ Local Variables တွေနှင့် Return Type တွေနေရာမှာ အသုံးပြုနိုင်ပါတယ်
- ဒါပေမဲ့ Wild Card တွေကို Generics Method Invocation တွေ၊ Generics Class instance creation တွေနှင့် Generic Type တွေရဲ့ Sub Type တွေမှာတော့ ရေးသား အသုံးပြုနိုင်မှာ မဟုတ်ပါဘူး

Without Using Wild Card



```
public static void main(String[] args) {  
    // this statement will be success  
    print(new Wrapper<Object>(10));  
  
    // all these statements will be compile error  
    print(new Wrapper<Integer>(10));  
    print(new Wrapper<String>("Hello"));  
}  
  
public static void print(Wrapper<Object> wrapper) {  
    System.out.println(wrapper.getData());  
}
```


Using Wild Card




```
public static void main(String[] args) {  
    // all these statements will be success  
    print(new Wrapper<Object>(10));  
    print(new Wrapper<Integer>(10));  
    print(new Wrapper<String>("Hello"));  
}  
  
// By using Wild Card in Type Parameter  
// any type parameter will be allowed  
public static void print(Wrapper<?> wrapper) {  
    System.out.println(wrapper.getData());  
}
```

Bounded Wild Card


- Generic Type တွေကို Declare လုပ်တုန်းက Type Parameter တွေမှာ Bounded လုပ်နိုင်ခဲ့သလို Generic Type တွေကို ပြန်ပြီးအသုံးပြုတဲ့အချိန် Wild Card Type တွေမှာလဲ Bounded လုပ်ပြီး အသုံးပြုနိုင်ပါတယ်
- Wild Card တွေကို Bounded လုပ်တဲ့အခါ Upper Bound ကိုလဲ အသုံးပြုနိုင်သလို Lower Bound ကိုလဲ အသုံးပြုနိုင်ပါတယ်
- Upper Bound လုပ်တဲ့အခါမှာ <? extends Type> လို့ရေးသားရမှာ ဖြစ်ပြီး Bounded လုပ်ထားတဲ့ Type နှင့် Sub Type တွေသာ အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်
- Lower Bound လုပ်တဲ့အခါမှာ <? super Type> လို့ရေးသားရမှာ ဖြစ်ပြီး Bounded လုပ်ထားတဲ့ Type နှင့် သူ့ရဲ့ Super Type တွေသာ အသုံးပြုလို့ရမှာ ဖြစ်ပါတယ်။

Upper Bounded Wild Card



```
public static void main(String[] args) {  
    // All these statements will be success  
    print(new Wrapper<Integer>(10));  
    print(new Wrapper<Number>(10));  
  
    // this statement will be compile error  
    // Because String is not a sub type of Number  
    print(new Wrapper<String>("Hello"));  
}  
  
public static void print(Wrapper<? extends Number> wrapper) {  
    System.out.println(wrapper.getData());  
}
```

Lower Bounded Wild Card



```
public static void main(String[] args) {  
    // All these statements will be success  
    print(new Wrapper<Integer>(10));  
    print(new Wrapper<Number>(10));  
  
    // this statement will be compile error  
    // Because String is not a super type of Number  
    print(new Wrapper<String>("Hello"));  
}  
  
public static void print(Wrapper<? super Integer> wrapper) {  
    System.out.println(wrapper.getData());  
}
```

Upper or Lower Bound

- Variable တွေကို အသုံးပြုပုံအပေါ်မူတည်ပြီး IN Type နှင့် OUT Type ဆိုပြီး ခွဲခြားနိုင်ပါတယ်
- Variable ထဲက တန်ဖိုးတွေကို ထုတ်ယူပြီး အသုံးပြုနေကြရရင် IN Type လို့သတ်မှတ်ပြီး၊ လက်ရှိ Program ထဲက Data တွေကို Variable ထဲကို ထည့်သွင်းနေတယ်ဆိုရင်တော့ OUT Type လို့ သတ်မှတ်လေ့ရှိပါတယ်
- IN Type Variable တွေက Data တွေကို Program ထဲမှာ အသုံးပြုရမှာ ဖြစ်လို့ ဘယ်လို Type မျိုးဖြစ်တယ်ဆိုတာကို သတ်မှတ်ထားဖို့လိုတဲအတွက် Wild Card တွေရဲ့ Bounded ကို စဉ်းစားတဲ့ နေရာမှာ Upper Bound ကို အသုံးပြုသင့်ပါတယ်
- Out Type Variable တွေကတော့ Program ထဲက Data တွေကို လက်ခံနိုင်ဖို့လိုတဲအတွက် Lower Bound ကို အသုံးပြုသင့်ပါတယ်

Type Erasure

- Type Erasure ဆိုတာကတော့ Java Compiler ရဲ့ Mechanism တစ်ခုဖြစ်ပြီး Generic Type တွေကို Compile လုပ်တဲ့ အခါမှာ အသုံးပြုပါတယ်
- တကယ်တမ်းကျတော့ Generic Type တွေရဲ့ Type Check Mechanism ဟာ Compile လုပ်တဲ့အခါမှာ ဆောင်ရွက်ကြ တာဖြစ်ပါတယ်
- Type Check ကို ဆောင်ရွက်ပြီးတဲ့အခါမှာ Type Erasure ကို အသုံးပြုပြီး Type Parameter တွေကို Raw Type အဖြစ် ပြောင်းလဲ ပေးပါတယ်
- Unbounded Type တွေကို Object အဖြစ်ပြောင်းပေးပြီး၊ Bounded Type တွေဆိုရင်တော့ Bounded လုပ်ထားတဲ့ Type အဖြစ်ပြောင်းပေးပါတယ်၊ လိုအပ်ပါက Bridge Methods တွေနှင့် Type Casting တွေကိုပါ ဆောင်ရွက်ပေးပါတယ်