16 Generics

Objectives

- To know the benefits of generics (§ 21.1).
- To use generic classes and interfaces (§ 21.2).
- To declare generic classes and interfaces (§ 21.3).
- To understand why generic types can improve reliability and readability (§ 21.3).
- To declare and use generic methods and bounded generic types (§ 21.4).
- To use raw types for backward compatibility (§ 21.5).
- To know wildcard types and understand why they are necessary (§ 21.6).
- To convert legacy code using JDK 1.5 generics (§ 21.7).
- To understand that generic type information is erased by the compiler and all instances of a generic class share the same runtime class file (§ 21.8).
- To know certain restrictions on generic types caused by type erasure (§ 21.8).
- To design and implement generic matrix classes (§ 21.9).

Why Do You Get a Warning?

```
public class ShowUncheckedWarning {
  public static void main(String[] args) {
    java.util.ArrayList list =
      new java.util.ArrayList();
    list.add("Java Programming");
  }
}
```

To understand the compile warning on this line, you need to learn JDK 1.5 generics.

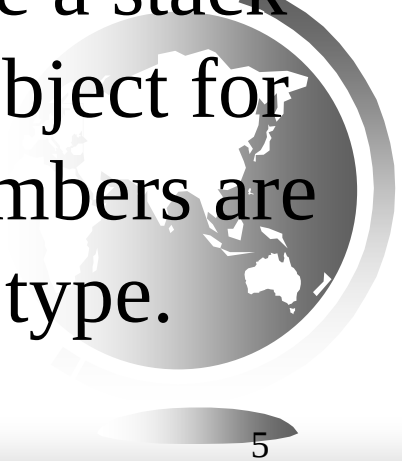
Fix the Warning

```
public class ShowUncheckedWarning {
  public static void main(String[] args) {
    java.util.ArrayList<String> list =
      new java.util.ArrayList<String>();
    list.add("Java Programming");
  }
}
```

No compile warning on this line.

What is Generics?

- *Generics* is the capability to **parameterize types**. With this capability, you can define a class or a method with generic types that can be substituted using concrete types by the **compiler**. For example, you may define a generic stack class that stores the elements of a generic type. From this generic class, you may create a stack object for holding strings and a stack object for holding numbers. Here, strings and numbers are concrete types that replace the generic type.



Why Generics?

- The key benefit of generics is to enable errors to be detected at compile time rather than at runtime.
- A generic class or method permits you to specify allowable types of objects that the class or method may work with. If you attempt to use the class or method with an incompatible object, the compile error occurs.

Generic Type

```
package java.lang;

public interface Comaprable {
  public int compareTo(Object o)
}
```

(a) Prior to JDK 1.5

```
package java.lang;

public interface Comaprable<T> {
  public int compareTo(T o)
}
```

(b) JDK 1.5

Generic Instantiation

Runtime error

```
Comparable c = new Date();
System.out.println(c.compareTo("red"));
```

(a) Prior to JDK 1.5

```
Comparable<Date> c = new Date();
System.out.println(c.compareTo("red"));
```

(b) JDK 1.5

Improves reliability

Compile error

Generic ArrayList in JDK 1.5

java.util.ArrayList
+ArrayList()
+add(o: Object) : void
+add(index: int, o: Object) : void
+clear(): void
+contains(o: Object): boolean
+get(index: int) : Object
+indexOf(o: Object) : int
+isEmpty(): boolean
+lastIndexOf(o: Object) : int
+remove(o: Object): boolean
+size(): int
+remove(index: int) : boolean
+set(index: int, o: Object) : Object

(a) ArrayList before JDK 1.5

java.util.ArrayList<E>
+ArrayList()
+add(o: E) : void
+add(index: int, o: E) : void
+clear(): void
+contains(o: Object): boolean
+get(index: int) : E
+indexOf(o: Object) : int
+isEmpty(): boolean
+lastIndexOf(o: Object) : int
+remove(o: Object): boolean
+size(): int
+remove(index: int) : boolean
+set(index: int, o: E) : E

(b) ArrayList in JDK 1.5

No Casting Needed

```
ArrayList<Double> list = new ArrayList<Double>();
list.add(5.5); // 5.5 is automatically converted to new Double(5.5)
list.add(3.0); // 3.0 is automatically converted to new Double(3.0)
Double doubleObject = list.get(0); // No casting is needed
double d = list.get(1); // Automatically converted to double
```

Declaring Generic Classes and Interfaces

GenericStack<E>	
-list: java.util.ArrayList<E>	An array list to store elements.
+GenericStack()	Creates an empty stack.
+getSize(): int	Returns the number of elements in this stack.
+peek(): E	Returns the top element in this stack.
+pop(): E	Returns and removes the top element in this stack.
+push(o: E): E	Adds a new element to the top of this stack.
+isEmpty(): boolean	Returns true if the stack is empty.

构造方法不是GenericStack<E>()

GenericStack

```java
public class GenericStack<E> {
  private java.util.ArrayList<E> list = new java.util.ArrayList<E>();

  public int getSize() {
    return list.size();
  }

  public E peek() {
    return list.get(getSize() - 1);
  }

  public void push(E o) {
    list.add(o);
  }

  public E pop() {
    E o = list.get(getSize() - 1);
    list.remove(getSize() - 1);
    return o;
  }

  public boolean isEmpty() {
    return list.isEmpty();
  }

  @Override
  public String toString() {
    return "stack: " + list.toString();
  }
}
```

Generic Methods

```
public static <E> void print(E[] list) {
  for (int i = 0; i < list.length; i++)
    System.out.print(list[i] + " ");
  System.out.println();
 }
```

```
public static void print(Object[] list) {
  for (int i = 0; i < list.length; i++)
    System.out.print(list[i] + " ");
  System.out.println();
 }
```

Generic Methods

- GenericMethodDemo.<Integer>print(integers);
- GenericMethodDemo.<String>print(strings);

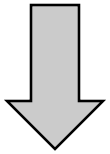
```
public class GenericMethodDemo {
  public static void main(String[] args ) {
    Integer[] integers = {1, 2, 3, 4, 5};
    String[] strings = {"London", "Paris", "New York", "Austin"};

    GenericMethodDemo.<Integer>print(integers);
    GenericMethodDemo.<String>print(strings);
  }

  public static <E> void print(E[] list) {
    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
    System.out.println();
  }
}
```

GenericMethodDemo

Generic Methods

- 若编译器能推断出所调用的方法，则可以省略。

```
GenericMethodDemo.<Integer>print(integers);
GenericMethodDemo.<String>print(strings);
```



```
GenericMethodDemo.print(integers);
GenericMethodDemo.print(strings);
```

Generic Methods

```
class ArrayAlg{
    public static <T> T getMiddle(T…a){
       return a[a.length/2];
   }
}
```

String middle = ArrayAlg. getMiddle(“Join”,”Q.”,”Public”);

String middle = ArrayAlg. getMiddle(3.14,1729,0); Compiler Error

编译器会将参数打包为1个Double，2个Integer对象，然后找这些类的共同超类，找到两个Number和Comparable。

Bounded Generic Type

```
public static void main(String[] args ) {
    Rectangle rectangle = new Rectangle(2, 2);
    Circle circle = new Circle (2);
    System.out.println("Same area? " + equalArea(rectangle, circle));
}

public static <E extends GeometricObject> boolean
       equalArea(E object1, E object2) {
    return object1.getArea() == object2.getArea();
}
```

根据参数，找共同的超类，
调用equalArea(GeometricOjbect o1, GeometricObject o2)
然后object.getArea()再是动态绑定

Raw Type and Backward Compatibility

```
// raw type
ArrayList list = new ArrayList();
```

This is roughly equivalent to

```
ArrayList<Object> list = new ArrayList<Object>();
```

Raw Type is Unsafe

```
// Max.java: Find a maximum object
public class Max {
  /** Return the maximum between two objects */
  public static Comparable max(Comparable o1, Comparable o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

Runtime Error:

Max.max("Welcome", 23);

Make it Safe

```
// Max1.java: Find a maximum object
public class Max1 {
  /** Return the maximum between two objects */
  public static <E extends Comparable<E>> E max(E o1, E o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

(1) 关键词extends而不是implements
<T extends BoundingType>
(2) 多个限定：T extends BType1 & BType2

根据参数，找共同的超类，
需要调用max(Comparable<String> o1, Comparable<Integer> o2)
显然，Comparable<String>与Comparable<Integer>没有关系

```
Max.max("Welcome", 23);
```

The inferred type Object&Serializable&Comparable<?> is not a valid substitute for the bounded parameter <E extends Comparable<E>>

Wildcards

Why wildcards are necessary? See this example.

WildCardDemo1

```
public class WildCardDemo1 {
  public static void main(String[] args ) {
    GenericStack<Integer> intStack = new GenericStack<Integer>();
    intStack.push(1); // 1 is autoboxed into new Integer(1)
    intStack.push(2);
    intStack.push(-2);

// Error:     System.out.print("The max number is " + max(intStack));
    System.out.print("The max number is " + max(intStack));
  }

  /** Find th
  public sta
    double r

    while (
      double
      if (value > max)
        max = value;
    }

    return max;
  }
}
```

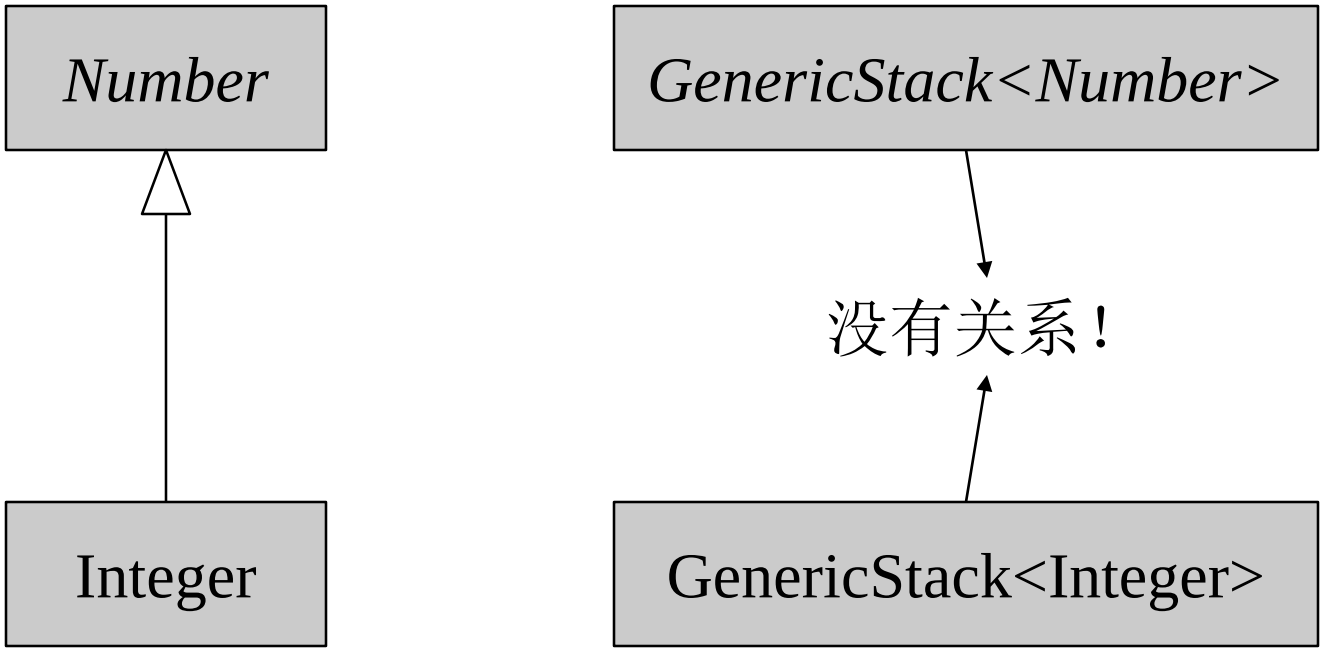
The method max(GenericStack<Number>) in the type WildCardDemo1 is not applicable for the arguments (GenericStack<Integer>)

4 quick fixes available:

- Change method 'max(GenericStack<Number>)' to 'max(GenericStack<Integer>)'
- Change to 'main(..)'
- Change type of 'intStack' to 'GenericStack<Number>'
- Create method 'max(GenericStack<Integer>)'

Press 'F2' for focus

GenericStack<Integer>与GenericStack<Number>并没有关系，只是Integer与Number有关系而已



若允许将GenericStack<Integer>转化为GenericStack<Number>

```
GenericStack<Interger> inS = new GenericStack<>();
GenericStack<Number> nS = inS;  //事实上illegal…
nS.push(1.23);
```

Wildcards

Why wildcards are necessary? See this example.

WildCardDemo1

? unbounded wildcard

? extends T bounded wildcard

? super T lower bound wildcard

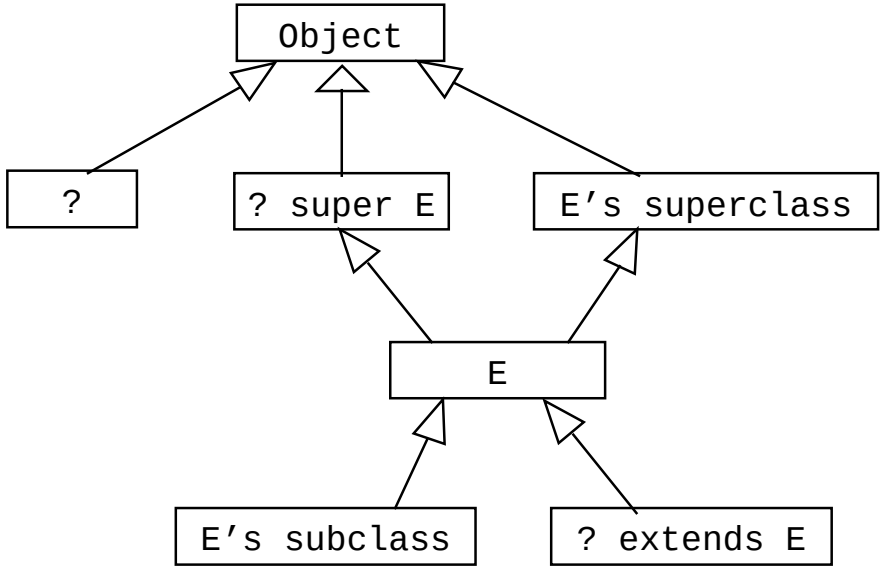
WildCardDemo2

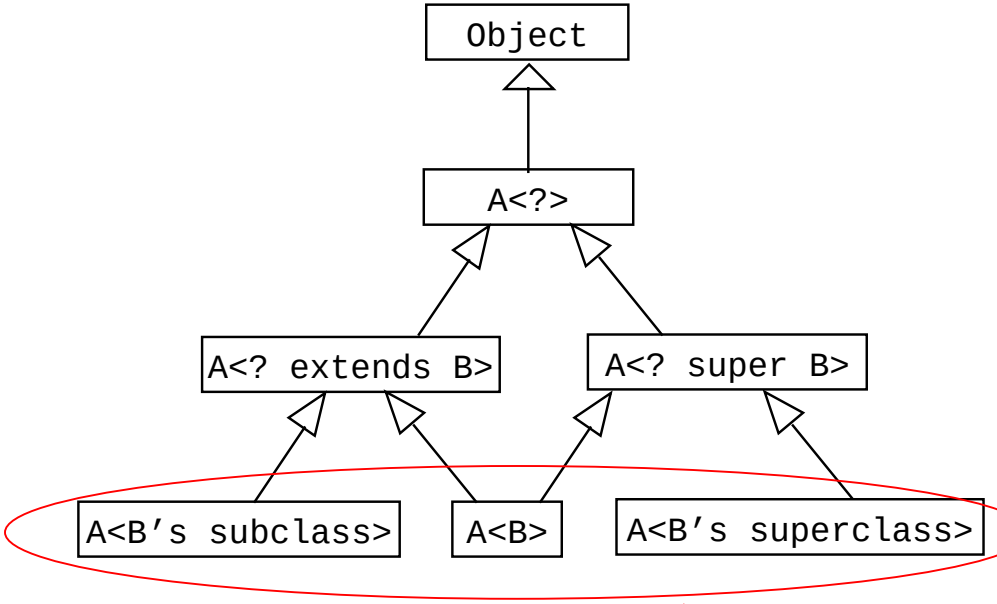
WildCardDemo3

```
public class WildCardDemo2 {
  public static void main(String[] args ) {
     GenericStack<Integer> intStack = new GenericStack<Integer>();
     intStack.push(1); // 1 is autoboxed into new Integer(1)
     intStack.push(2);
     intStack.push(-2);

     print(intStack);
  }

  /** Print objects and empties the stack */
  public static void print(GenericStack<?> stack) {
    while (!stack.isEmpty()) {
      System.out.print(stack.pop() + " ");
    }
  }
}
```

```java
public class WildCardDemo3 {
  public static void main(String[] args) {
    GenericStack<String> stack1 = new GenericStack<String>();
    GenericStack<Object> stack2 = new GenericStack<Object>();
    stack2.push("Java");
    stack2.push(2);
    stack1.push("Sun");
    add(stack1, stack2);
    WildCardDemo2.print(stack2);
  }

  public static <T> void add(GenericStack<T> stack1, GenericStack<? super T> stack2) {
    while (!stack1.isEmpty())
      stack2.push(stack1.pop());
  }
}
```

Generic Types and Wildcard Types





无关系

注意：参数化类型**没有实际类型参数的继承关系！**

List<Integer> list = new List<Object>(); //**报错，反之亦然（也是错的）**

泛型的继承关系需要通过**通配符**

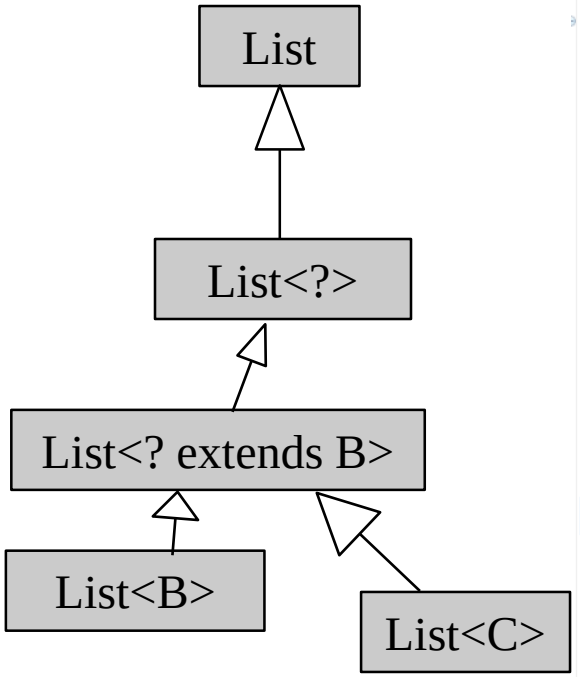
JDK7之后的新特性：后一个泛型可以省略，JDK可以自动推断！

25. Given code below:

```
class B {}
class C extends B {}
```

Which function below is NOT able to accept objects of both List<B> and List<C>?

A. `void f(List<? extends B> list);`
B. `void f(List<B> list);`
C. `void f(List list);`
D. `void f(List<?> list);`




```
public static void main(String[] args) {
    GenericTest gt = new GenericTest();
    List<B> b = new ArrayList<B>();
    List<C> c = new ArrayList<C>();
    gt.f1(b);
    gt.f1(c);

    gt.f2(b);
    gt.f2(c);

}

cl
class C extends B{}
void f1(List<? extends B> list){}
void f2(List<B> list){}
void f3(List list){}
void f4(List<?> list){}

}
```

The method f2(List<GenericTest.B>) in the type GenericTest is not applicable for the arguments (List<GenericTest.C>)

6 quick fixes available:

- Change method 'f2(List<B>)' to 'f2(List<C>)'
- Change to 'f1(..)'
- Change to 'f3(..)'
- Change to 'f4(..)'
- Change type of 'c' to 'List<B>'
- Create method 'f2(List<C>)'

Avoiding Unsafe Raw Types

Use

new ArrayList<ConcreteType>()

Instead of

new ArrayList();

TestArrayListNew

Run

Erasure and Restrictions on Generics

Generics are implemented using an approach called *type erasure*. The compiler uses the generic type information to compile the code, but erases it afterwards. So the generic information is not available at run time. This approach enables the generic code to be backward-compatible with the legacy code that uses raw types.

Compile Time Checking

For example, the compiler checks whether generics is used correctly for the following code in (a) and translates it into the equivalent code in (b) for runtime use. The code in (b) uses the raw type.

```
ArrayList<String> list = new ArrayList<String>();
list.add("Oklahoma");
String state = list.get(0);
```

(a)

```
ArrayList list = new ArrayList();
list.add("Oklahoma");
String state = (String)(list.get(0));
```

(b)

Important Facts

It is important to note that a generic class is shared by all its instances regardless of its actual generic type.

```
GenericStack<String> stack1 = new GenericStack<String>();
GenericStack<Integer> stack2 = new GenericStack<Integer>();
```

Although GenericStack<String> and GenericStack<Integer> are two types, but there is only one class GenericStack loaded into the JVM.

```java
public class GenericStack<E> {
  private java.util.ArrayList<E> list = new java.util.ArrayList<E>();

  public int getSize() {
    return list.size();
  }

  public E peek() {
    return list.get(getSize() - 1);
  }

  public void push(E o) {
    list.add(o);
  }

  public E pop() {
    E o = list.get(getSize() - 1);
    list.remove(getSize() - 1);
    return o;
  }
```

类型变量被替换为限定类型（无限定的变量用Object)

E没有限定，则替换为Object

擦除泛型后的原始类型

```
public class GenericStack{
  private java.util.ArrayList list = …
  public Object peek()…
  public void push(Object o)…
  public Object pop()…
  …
}
```

Important Facts

- ArrayList<String> list1 = new ArrayList<String>();
- ArrayList<Integer> list2 = new ArrayList<Integer>();

- System.out.println(list1 instanceof ArrayList);
- System.out.println(list2 instanceof ArrayList);

- 表达式list1 instanceof ArrayList<String>是错误的。ArrayList<String>并没有在JVM中存储为一个类。

Restrictions on Generics

- Restriction 1: Cannot Create an Instance of a Generic Type. (i.e., new E()).

  E object = new E(); //Error

类型擦除后，E就变为Object了，本意也不希望调研new Object

Restrictions on Generics

- Restriction 2: Generic Array Creation is Not Allowed. (i.e., new E[100]).

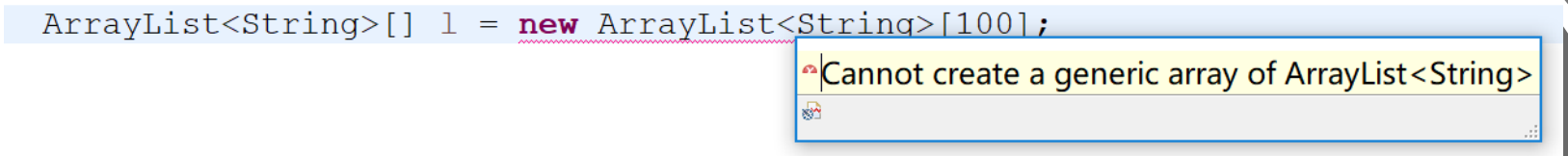
E[] elements = new E[100];

用E[] elements = (E[]) new Object[100]来规避，但有一个编译警告

例如：

不允许用泛型类型来创建泛型数组：

ArrayList<String>[] list = new ArrayList<String>[100];



ArrayList<String>[]list = (ArrayList<String>[])new ArrayList[100]; //可行

List<String>[] array = new ArrayList<String>[]; （如果可行）

Object[] objectArray = array;

objectArray[0] = 1;

假设第1行代码是合法的，下面的3行代码都是没有问题的。objectArray数组的0号实际保存的是一个整型对象，而在第4行代码出却要转换为String类型，这会发生ClassCastException

```
27          ArrayList<String>[] l = (ArrayList<String>[])new ArrayList[100];
28          Object[]o = l;
29          o[0] = 1;
30          String s = l[0].get(0);
31          System.out.println(s);
```

Problems | Javadoc | Declaration | Console

```
<terminated> Test1 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_141\jre\bin\javaw.exe (2018年11月27
Exception in thread "main" java.lang.ArrayStoreException: java.lang.Integer
        at test.Test1.main(Test1.java:29)
```

```
ArrayList<String>[] l = (ArrayList<String>[])new ArrayList[100];
Object[]o = l;
((ArrayList<String>)o[0]).add("S");
String s = l[0].get(0);
System.out.println(s);
```

```
<terminated> Test1 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_141\jre\bi
Exception in thread "main" java.lang.NullPointerException
        at test.Test1.main(Test1.java:29)
```

```
ArrayList<String>[] l = (ArrayList<String>[])new ArrayList[100];
l[0] = new ArrayList<String>();
Object[]o = l;
((ArrayList<String>)o[0]).add("S");
String s = l[0].get(0);
System.out.println(s);
```

Restrictions on Generics

- Restriction 3: A Generic Type Parameter of a Class Is Not Allowed in a Static Context.

泛型类的所有实例都有相同的运行时类，所以泛型类的静态变量和方法是被它的所有实例共享的。所以在静态方法、数据域或初始化语句中，为了类而引用泛型参数是非法的。

```
public class Test<E>{
    public static void m(E o1){ // illegal
    }
    static {
        E o2;
    }
}
```

由于泛型的具体参数要在实例化是才能确定，而静态变量和静态方法无需实例化就可以调用。当对象都还没有创建时，就调用与泛型变量相关的方法，当然是错误的。

Restrictions on Generics

- Restriction 4: Exception Classes Cannot be Generic.

  泛型类不能扩展java.lang.Throwable。

```
public class MyException<T> extends Exception{
}
```

JVM必须检查try子句中抛出的异常类型是否与catch子句中指定的类型匹配。但在运行期间类型信息不知道。

```
try{
……
}
catch(MyException<T> ex){
}
```

```
try {
   doSomeStuff();
} catch (SomeException<Integer> e) {
   // ignore that
} catch (SomeException<String> e) {
   crashAndBurn()
}
```

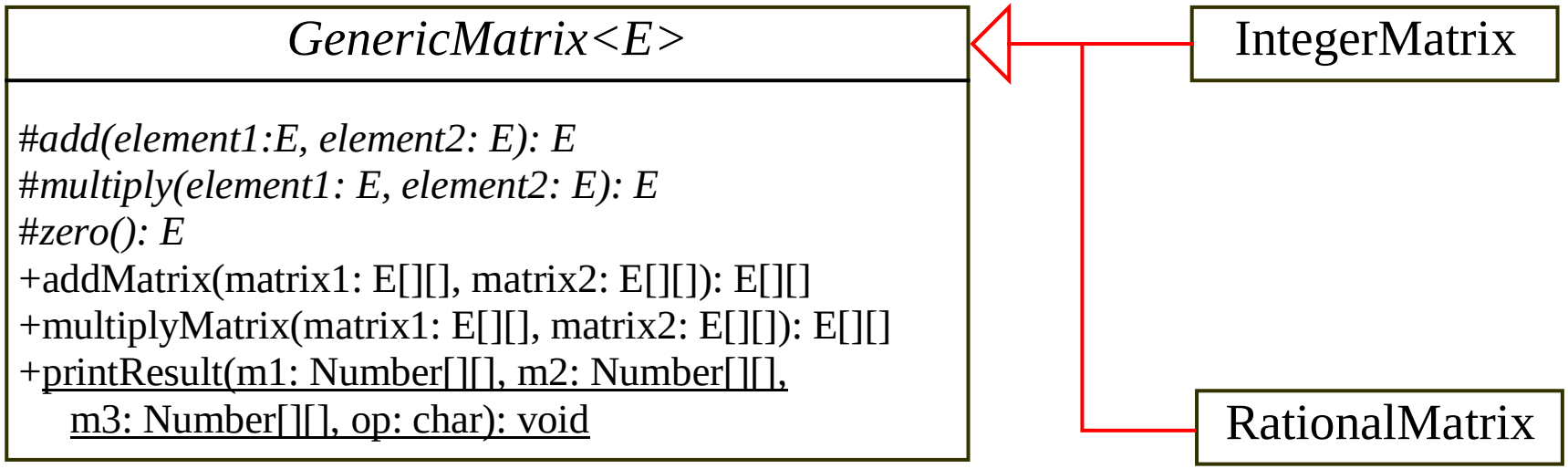
- Both SomeException<Integer> and SomeException<String> are erased to the same type, there is no way for the JVM to distinguish the exception instances, and therefore no way to tell which catch block should be executed.

Designing Generic Matrix Classes

- Objective: This example gives a generic class for matrix arithmetic. This class implements matrix addition and multiplication common for all types of matrices.

GenericMatrix

UML Diagram


GenericMatrix<E>
#add(element1:E, element2: E): E
#multiply(element1: E, element2: E): E
#zero(): E
+addMatrix(matrix1: E[][], matrix2: E[][]): E[][]
+multiplyMatrix(matrix1: E[][], matrix2: E[][]): E[][]
+printResult(m1: Number[][], m2: Number[][],
m3: Number[][], op: char): void
IntegerMatrix
RationalMatrix

Source Code

- Objective: This example gives two programs that utilize the GenericMatrix class for integer matrix arithmetic and rational matrix arithmetic.

IntegerMatrix

TestIntegerMatrix

Run

RationalMatrix

TestRationalMatrix

Run

```java
public abstract class GenericMatrix<E extends Number> {
  /** Abstract method for adding two elements of the matrices */
  protected abstract E add(E o1, E o2);

  /** Abstract method for multiplying two elements of the matrices */
  protected abstract E multiply(E o1, E o2);

  /** Abstract method for defining zero for the matrix element */
  protected abstract E zero();

  /** Add two matrices */
  public E[][] addMatrix(E[][] matrix1, E[][] matrix2) {
    // Check bounds of the two matrices
    if ((matrix1.length != matrix2.length) ||
        (matrix1[0].length != matrix2[0].length)) {
      throw new RuntimeException(
        "The matrices do not have the same size");
    }

    E[][] result =
      (E[][])new Number[matrix1.length][matrix1[0].length];

    // Perform addition
    for (int i = 0; i < result.length; i++)
      for (int j = 0; j < result[i].length; j++) {
        result[i][j] = add(matrix1[i][j], matrix2[i][j]);
      }

    return result;
  }

  /** Multiply two matrices */
  public E[][] multiplyMatrix(E[][] matrix1, E[][] matrix2) {
    // Check bounds
    if (matrix1[0].length != matrix2.length) {
      throw new RuntimeException(
        "The matrices do not have compatible size");
    }

    // Create result matrix
```

```java
public class IntegerMatrix extends GenericMatrix<Integer> {
  @Override /** Add two integers */
  protected Integer add(Integer o1, Integer o2) {
    return o1 + o2;
  }

  @Override /** Multiply two integers */
  protected Integer multiply(Integer o1, Integer o2) {
    return o1 * o2;
  }

  @Override /** Specify zero for an integer */
  protected Integer zero() {
    return 0;
  }
```

```java
public class RationalMatrix extends GenericMatrix<Rational> {
  @Override /** Add two rational numbers */
  protected Rational add(Rational r1, Rational r2) {
    return r1.add(r2);
  }

  @Override /** Multiply two rational numbers */
  protected Rational multiply(Rational r1, Rational r2) {
    return r1.multiply(r2);
  }

  @Override /** Specify zero for a Rational number */
  protected Rational zero() {
    return new Rational(0,1);
  }
}
```