Chapter 19 Generics

CS165
Colorado State University

Original slides by Daniel Liang
Modified slides by Wim Bohm, Sudipto Ghosh

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, a compile error occurs.

What is Generics?

Generics is the capability to parameterize types.

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 collection class that stores the elements of a generic type. From this generic class, you may create a collection for holding Strings and a collection for holding Floats.

Strings and Floats are concrete types that replace the generic type.

Raw Type

```
package java.lang;

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

(a) Prior to JDK 1.5

Generic Type

```
package java.lang;

public interface Comparable<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

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");
  }
}
```

Compile time warning on this line, you are using a “raw” type.

Fix the Warning

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

No compile warning on this line.

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

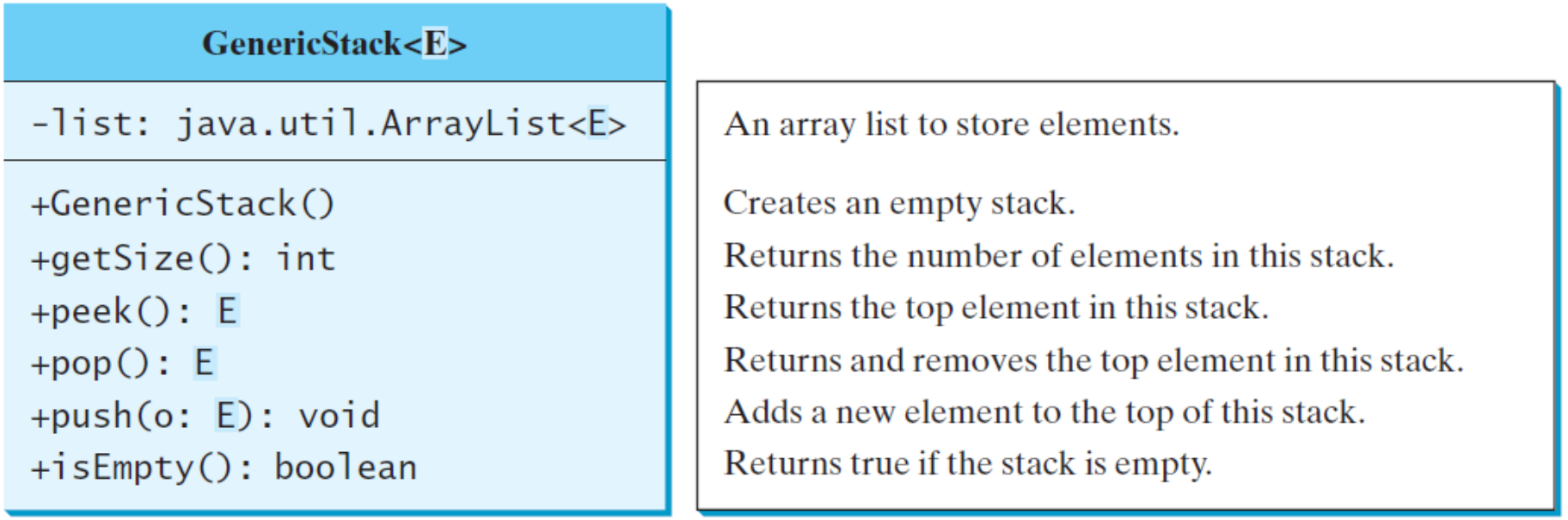
java.util.ArrayList<E>
+ArrayList() +add(o: E): void +add(index: int, o: E): void +clear(): void +contains(o: Object): boolean +get(index:int): E +indexOf(o: Object): int +isEmpty(): boolean +lastIndexOf(o: Object): int +remove(o: Object): boolean +size(): int +remove(index: int): boolean +set(index: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

(b) ArrayList since JDK 1.5

No Casting Needed

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

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 main(String[] args){
Integer[] integers = {1,2,3,4,5};
String[] strings = {‘apples”, “bananas”, “coconuts”};
print(integers);
print (strings);
}
```

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();
}
```

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.

The generic information is not available at run time.

This enables the generic code to be backward-compatible with the legacy code that uses raw types.

Shared generic class

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

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

Restrictions on Generics

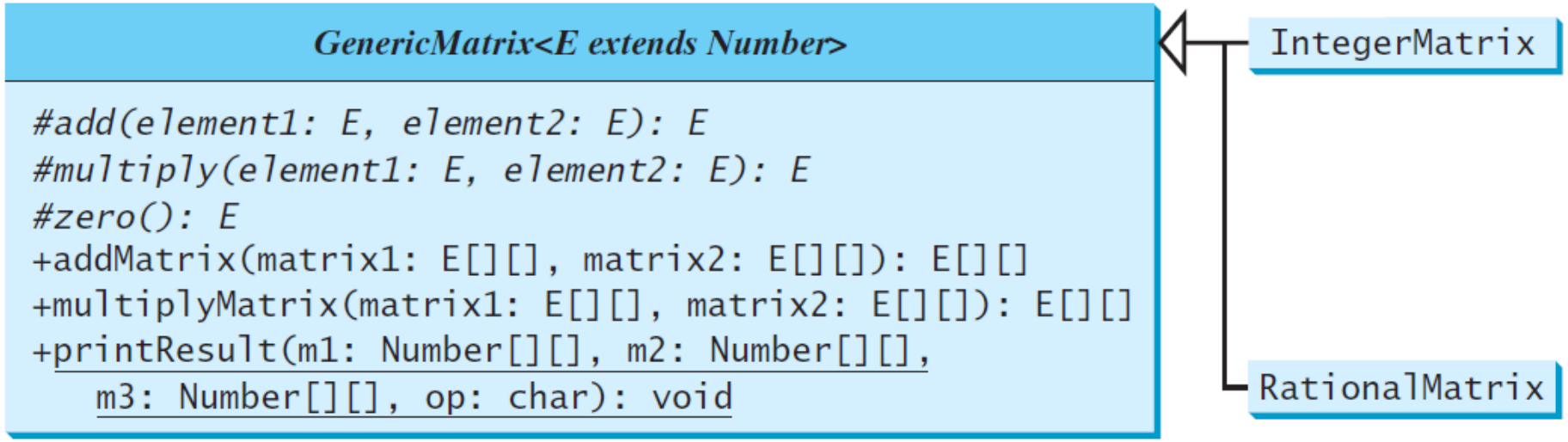
- Restriction 1: Cannot Create an Instance of a Generic Type. (i.e., new E()).
- Restriction 2: Generic Array Creation is Not Allowed. (i.e., new E[100]).
- Restriction 3: A Generic Type Parameter of a Class Is Not Allowed in a Static Context.
- Restriction 4: Exception Classes Cannot be Generic.

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 extends Number>
#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