Java Generics

Parametric Polymorphism

CASE STUDY: GENERIC MATRIX

DR. ERIC CHOU

IEEE SENIOR MEMBER

Case Study: Generic Matrix Class I

This lecture presents a case study on designing classes for matrix operations using generic types.

- The addition and multiplication operations for all matrices are similar except that their element types differ. Therefore, you can design a superclass that describes the common operations shared by matrices of all types regardless of their element types, and you can define subclasses tailored to specific types of matrices.

- This case study gives implementations for two types: **int** and **Rational**. For the **int** type, the wrapper class **Integer** should be used to wrap an **int** value into an object, so that the object is passed in the methods for operations.

Case Study: Generic Matrix Class II

This lecture presents a case study on designing classes for matrix operations using generic types.

- The class diagram is shown in Figure E. The methods **addMatrix** and **multiplyMatrix** add and multiply two matrices of a generic type **E[][]**. The static method **printResult** displays the matrices, the operator, and their result. The methods **add**, **multiply**, and **zero** are abstract, because their implementations depend on the specific type of the array elements. For example, the **zero()** method returns **0** for the **Integer** type and **0/1** for the **Rational** type. These methods will be implemented in the subclasses in which the matrix element type is specified.

UML GenericMatrix class

The GenericMatrix class is an abstract superclass for IntegerMatrix and RationalMatrix

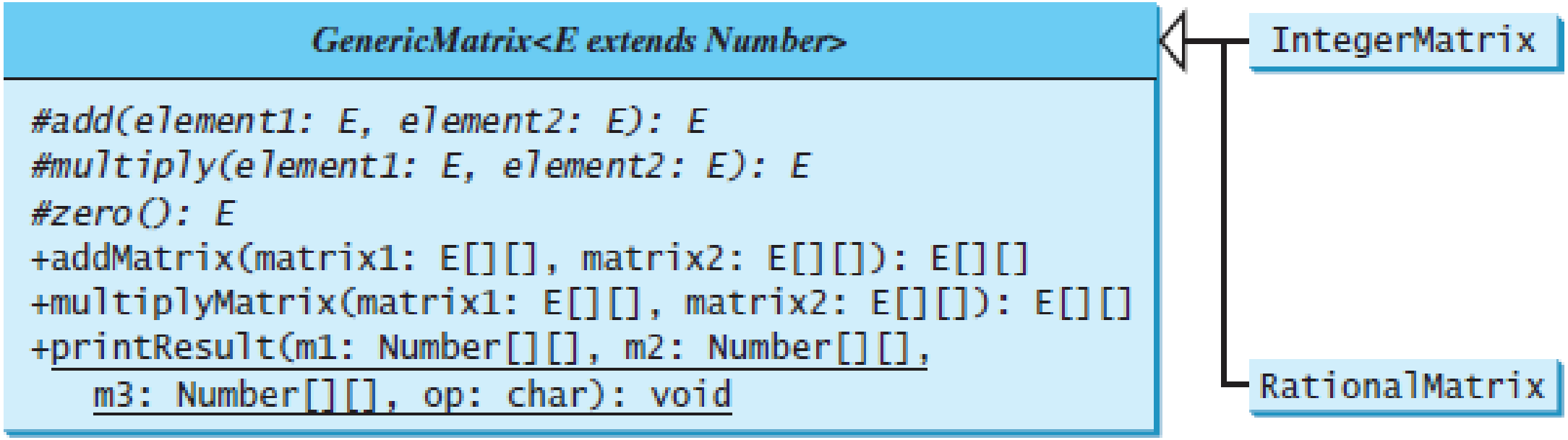

Figure E.

Case Study: Generic Matrix Class I

- **IntegerMatrix** and **RationalMatrix** are concrete subclasses of **GenericMatrix**. These two classes implement the **add**, **multiply**, and **zero** methods defined in the **GenericMatrix** class.
- **GenericMatrix.java** implements the **GenericMatrix** class. **<E extends Number>** in line 1 specifies that the generic type is a subtype of **Number**. Three abstract methods—**add**, **multiply**, and **zero**—are defined in lines 3, 6, and 9. These methods are abstract because we cannot implement them without knowing the exact type of the elements. The **addMaxtrix** (lines 12–30) and **multiplyMatrix** (lines 33–57) methods implement the methods for adding and multiplying two matrices. All these methods must be nonstatic, because they use generic type **E** for the class. The **printResult** method (lines 60–84) is static because it is not tied to specific instances.

Case Study: Generic Matrix Class II

- The matrix element type is a generic subtype of **Number**. This enables you to use an object of any subclass of **Number** as long as you can implement the abstract **add**, **multiply**, and **zero** methods in subclasses.
- The **addMatrix** and **multiplyMatrix** methods (lines 12–57) are concrete methods. They are ready to use as long as the **add**, **multiply**, and **zero** methods are implemented in the subclasses.
- The **addMatrix** and **multiplyMatrix** methods check the bounds of the matrices before performing operations. If the two matrices have incompatible bounds, the program throws an exception (lines 16, 36).

Case Study: Generic Matrix Class III

- IntegerMatrix.java implements the **IntegerMatrix** class. The class extends **GenericMatrix<Integer>** in line 1. After the generic instantiation, the **add** method in **GenericMatrix<Integer>** is now **Integer add(Integer o1, Integer o2)**. The **add**, **multiply**, and **zero** methods are implemented for **Integer** objects. These methods are still protected, because they are invoked only by the **addMatrix** and **multiplyMatrix** methods.

Case Study: Generic Matrix Class IV

- RationalMatrix.java implements the **RationalMatrix** class. The **Rational** class was introduced in Rational.java. **Rational** is a subtype of **Number**. The **RationalMatrix** class extends **GenericMatrix<Rational>** in line 1. After the generic instantiation, the **add** method in **GenericMatrix<Rational>** is now **Rational add(Rational r1, Rational r2)**. The **add**, **multiply**, and **zero** methods are implemented for **Rational** objects. These methods are still protected, because they are invoked only by the **addMatrix** and **multiplyMatrix** methods.

Case Study: Generic Matrix Class V

TestIntegerMatrix.java gives a program that creates two **Integer** matrices (lines 4–5) and an **IntegerMatrix** object (line 8), and adds and multiplies two matrices in lines 12 and 16.

TestRationalMatrix gives a program that creates two **Rational** matrices (lines 4–10) and a **RationalMatrix** object (line 13) and adds and multiplies two matrices in lines 17 and 19.

Case Study: GenericMatrix.java

Go BlueJ!