Generics

CSE219, Computer Science III

Stony Brook University

http://www.cs.stonybrook.edu/~cse219

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");
  }

}
```

It is better to get a compiling warning than a runtime error!

(c) Paul Fodor & Pearson Inc.

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");
    list.add(new Number(1));
  }
}
```

No compile warning on this line.

(c) Paul Fodor & Pearson Inc.

What are 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.
- 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.

    - **Strings and numbers are concrete types that replace the generic type.**

(c) Paul Fodor & Pearson Inc.

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.

(c) Paul Fodor & Pearson Inc.

Generic Types

```
package java.lang;

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

(a) Prior to JDK 1.5

```
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

Compile error

Improves reliability

(c) Paul Fodor & Pearson Inc.

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

(c) Paul Fodor & Pearson Inc.

Advantage: 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
             // Property of numeric wrapper classes
```


(c) Paul Fodor & Pearson Inc.

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.

(c) Paul Fodor & Pearson Inc.

Declaring Generic Classes and Interfaces

```
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    // Java annotation: also used at compile time to
    public String toString() {      // detect override errors
        return "stack: " + list.toString();
    }
}
```

GenericStack.java

(c) Paul Fodor & Pearson Inc.

Declaring Generic Classes and Interfaces

```
public static void main(String[] args){
     GenericStack<Integer> s1;
     s1 = new GenericStack<>();
     s1.push(1);
     s1.push(2);
     System.out.println(s1);
     GenericStack<String> s2 = new GenericStack<>();
     s2.push("Hello");
     s2.push("World");
     System.out.println(s2);
 }
```

Output:
stack: [1, 2]
stack: [Hello, World]

(c) Paul Fodor & Pearson Inc.

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) {
  String[] s3 = {"Hello", "again"};
  print(s3);
}
```

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


(c) Paul Fodor & Pearson Inc.

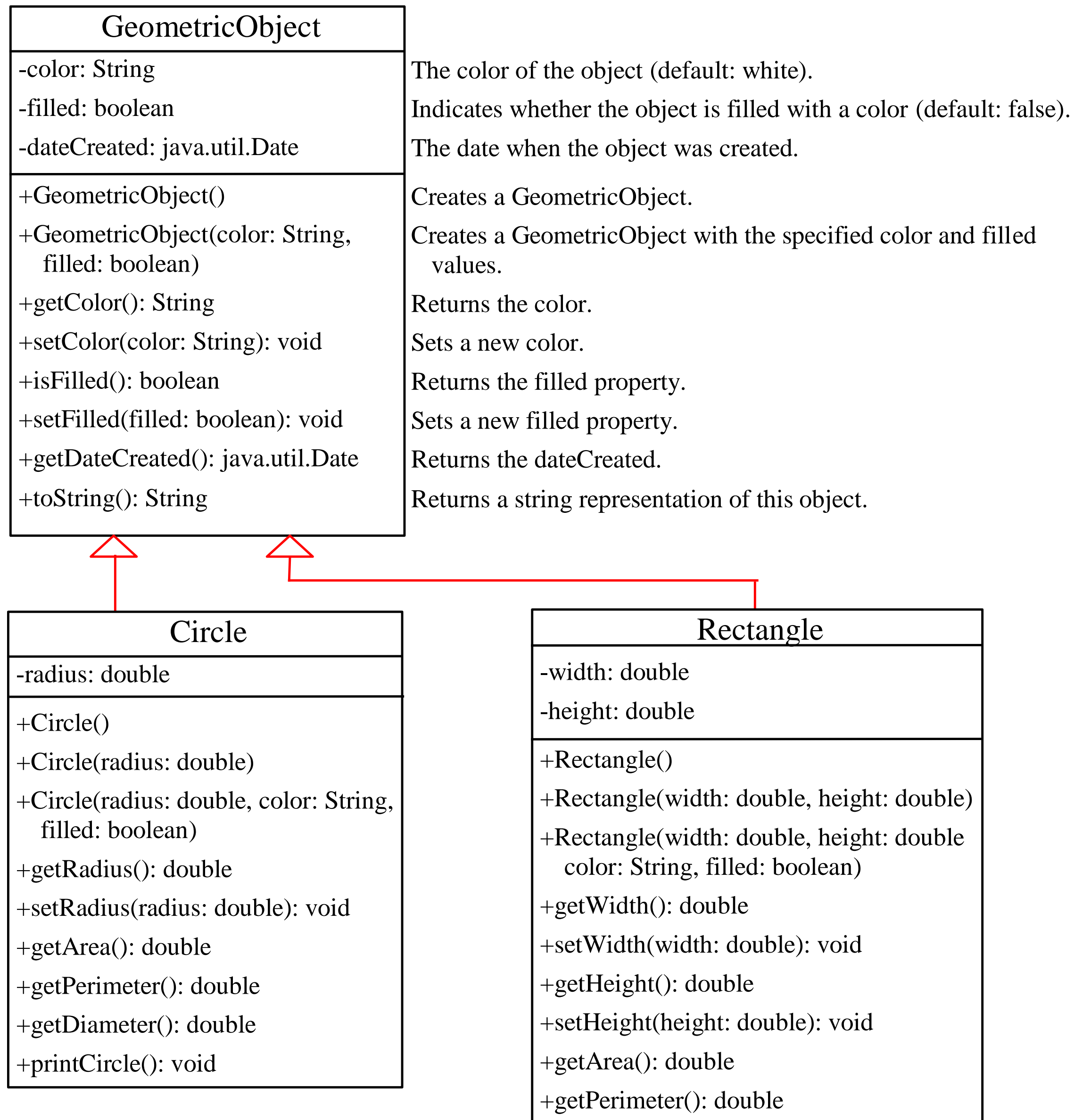
Bounded Generic Type

- Consider Circle and Rectangle extend GeometricObject

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

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));
}
```


(c) Paul Fodor & Pearson Inc.

Superclasses and Subclasses



(c) Paul Fodor & Pearson Inc.

```java
public abstract class GeometricObject {
  private String color = "white";
  private boolean filled;
  private java.util.Date dateCreated;
  protected GeometricObject() {
    dateCreated = new java.util.Date();
  }
  protected GeometricObject(String color, boolean filled) {
    dateCreated = new java.util.Date();
    this.color = color;
    this.filled = filled;
  }
  public String getColor() {   return color;   }
  public void setColor(String color) {  this.color = color;   }
  public boolean isFilled() {   return filled;   }
  public void setFilled(boolean filled) {  this.filled = filled;   }
  public java.util.Date getDateCreated() {   return dateCreated;   }
  public String toString() {
    return "created on " + dateCreated + "\ncolor: " + color +
      " and filled: " + filled;
  }
  /** Abstract method getArea */
  public abstract double getArea();
  /** Abstract method getPerimeter */
  public abstract double getPerimeter();
```


(c) Paul Fodor & Pearson Inc.

```
public class Circle extends GeometricObject {
  private double radius;
  public Circle() {    }
  public Circle(double radius) {
    this.radius = radius;
  }
  public double getRadius() {
    return radius;
  }
  public void setRadius(double radius) {
    this.radius = radius;
  }
  public double getArea() {
    return radius * radius * Math.PI;
  }
  public double getDiameter() {
    return 2 * radius;
  }
  public double getPerimeter() {
    return 2 * radius * Math.PI;
  }
  /* Print the circle info */
  public void printCircle() {
    System.out.println("The circle is created " + getDateCreated() +
      " and the radius is " + radius);
```


(c) Paul Fodor & Pearson Inc.

```
}
```

```
public class Rectangle extends GeometricObject {
  private double width;
  private double height;
  public Rectangle() {   }
  public Rectangle(double width, double height, String color,
                boolean filled) {
    super(color,filled);
    this.width = width;
    this.height = height;
  }
  public double getWidth() {    return width;   }
  public void setWidth(double width) {      this.width = width;   }
  public double getHeight() {    return height;   }
  public void setHeight(double height) {     this.height = height;   }

  public double getArea() {
    return width * height;
  }

  public double getPerimeter() {
    return 2 * (width + height);
  }
}
```


(c) Paul Fodor & Pearson Inc.

Raw Type and Backward Compatibility

- In JDK1.5 and higher, the ***Raw type:***

  ArrayList list = new ArrayList();

  - This is roughly equivalent to:

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

  - Therefore, all programs written in previous JDK versions are still executable.
- ***Backward compatibility***: a technology is backward compatible if it can work with input generated for an older technology.
  - Example: Python 3.0 broke backward compatibility to Python 2.6
  - Moreover, the new features (like dynamic typing) made mechanical translation from Python 2.x to Python 3.0 very difficult
- ***Forward compatibility*** implies that old devices allow input formats of new devices to run (without supporting the new features).
  - Example: edits to the Python 3.0 code were discouraged for so long as the code needed to run on Python 2.x

(c) Paul Fodor & Pearson Inc.

Raw Type is Unsafe

```
// Return the maximum between two objects
public static Comparable max1(Comparable o1,
         Comparable o2) {
  if (o1.compareTo(o2) > 0)
    return o1;
  else
    return o2;
}
```

- max1("Welcome", 23);
  - No compile error.
  - Runtime Error!

(c) Paul Fodor & Pearson Inc.

Make it Safe

```
// Return the maximum between two objects
public static <E extends Comparable<E>>
        E max2(E o1, E o2) {
  if (o1.compareTo(o2) > 0)
    return o1;
  else
    return o2;
}
```

- max2("Welcome", 23);
  - Compiler Error!

(c) Paul Fodor & Pearson Inc.

Wildcards

- ? unbounded wildcard
- ? extends T bounded wildcard
- ? super T lower bound wildcard

(c) Paul Fodor & Pearson Inc.

```
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);
        System.out.print("The max number is " + max(intStack));
               // Compiling Error: max cannot be applied to
    }          // GenericStack<Integer>; It expects GenericStack<Number>

    /**
     * Find the maximum in a stack of numbers
     */
    public static double max(GenericStack<Number> stack) {
        double max = stack.pop().doubleValue(); // initialize max
        while (!stack.isEmpty()) {
            double value = stack.pop().doubleValue();
            if (value > max) {
                max = value;
            }
        }
        return max;
    }
}
```


(c) Paul Fodor & Pearson Inc.

```java
public class WildCardDemo1B {
    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);
        System.out.print("The max number is " + max(intStack));
    }

    /**
     * Find the maximum in a stack of numbers
     */
    public static double max(GenericStack<? extends Number> stack) {
        double max = stack.pop().doubleValue(); // initialize max
        while (!stack.isEmpty()) {
            double value = stack.pop().doubleValue();
            if (value > max) {
                max = value;
            }
        }
        return max;
    }
}
```

Output: The max number is 2.0

(c) Paul Fodor & Pearson Inc.

```
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() + " ");
        }
    }
}
```

Output: -2 2 1

(c) Paul Fodor & Pearson Inc.

```
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(8);
        stack1.push("Oracle");
        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());
        }
    }
}
```

Output: Oracle 8 Java

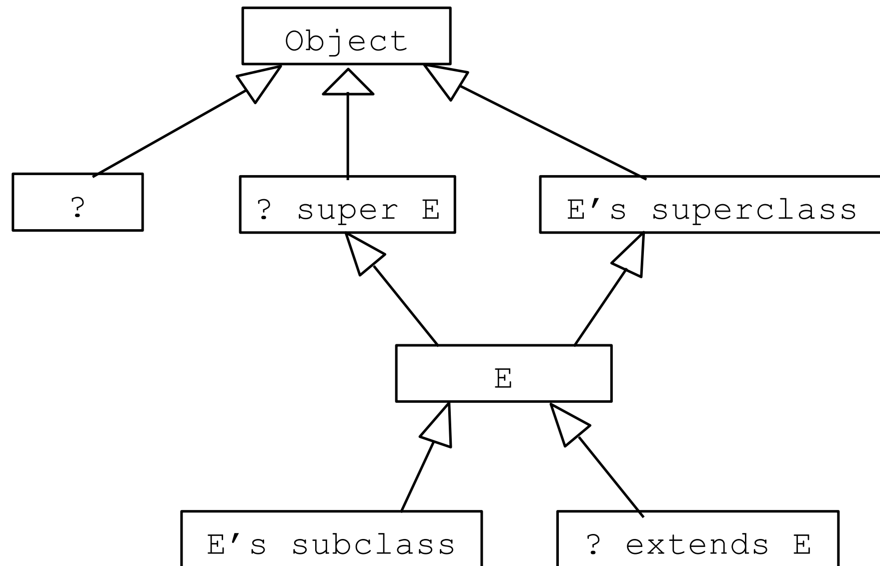
(c) Paul Fodor & Pearson Inc.

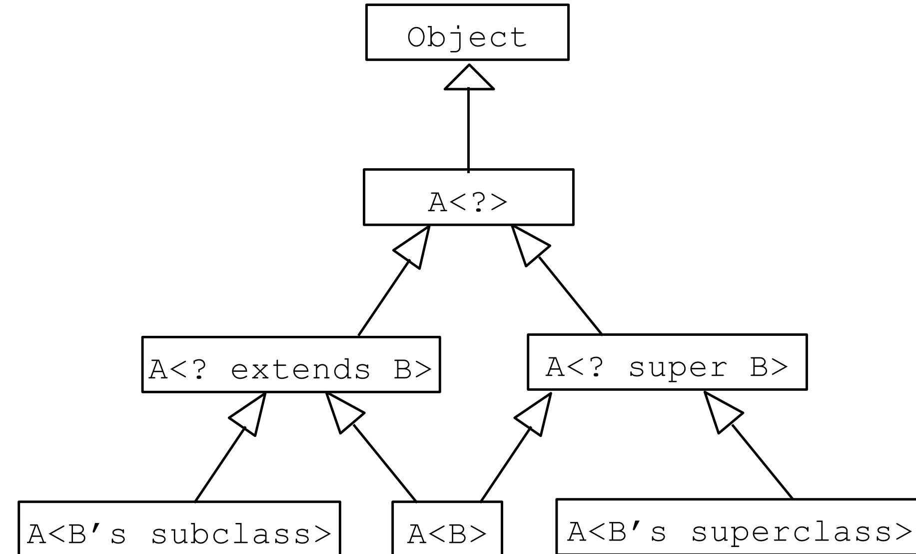
```
public class AnyWildCardDemo {

    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);
        print(intStack);
    }

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

Output: 2 1

(c) Paul Fodor & Pearson Inc.

Generic Types and Wildcard Types





(c) Paul Fodor & Pearson Inc.

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.

(c) Paul Fodor & Pearson Inc.

Compile Time Checking

- 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)

(c) Paul Fodor & Pearson Inc.

Erasure and Restrictions on Generics

- 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.

(c) Paul Fodor & Pearson Inc.

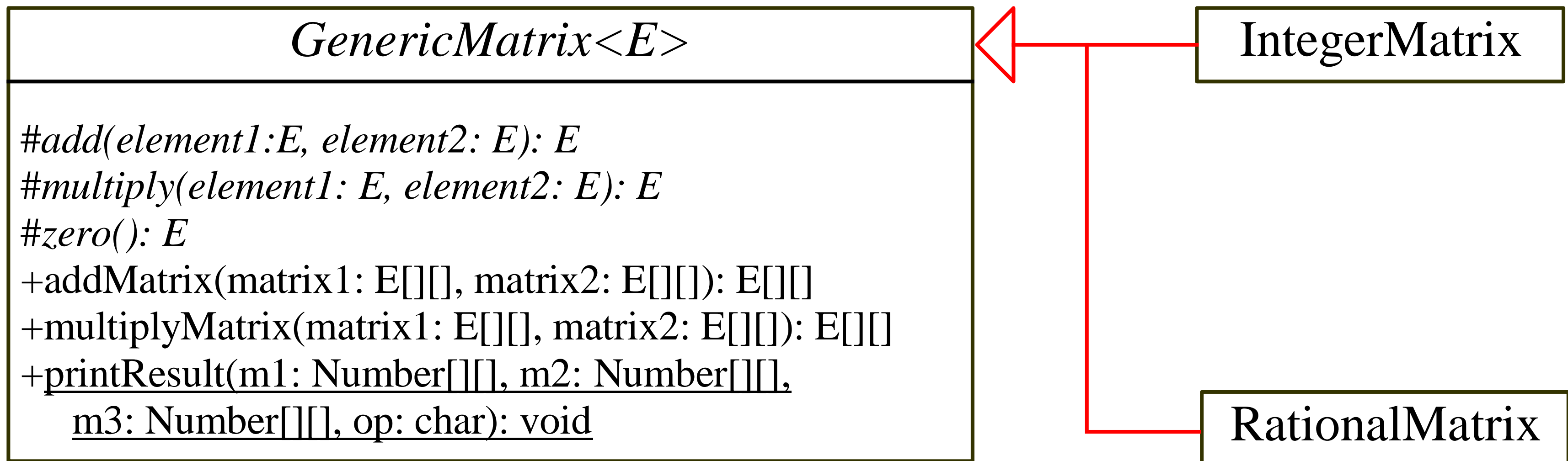
Restrictions on Generics

- 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 (i.e., static E x)
- Restriction 4: Exception Classes Cannot be Generic.

(c) Paul Fodor & Pearson Inc.

Designing Generic Matrix Classes

- Example: A generic class for matrix arithmetic.
  - The class implements matrix addition and multiplication common for all types of matrices.



(c) Paul Fodor & Pearson Inc.

```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();
```


(c) Paul Fodor & Pearson Inc.

```java
/**
 * 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;
}
```


(c) Paul Fodor & Pearson Inc.

```
/**
 * 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
    E[][] result =
            (E[][]) new Number[matrix1.length][matrix2[0].length];

    // Perform multiplication of two matrices
    for (int i = 0; i < result.length; i++) {
        for (int j = 0; j < result[0].length; j++) {
            result[i][j] = zero();

            for (int k = 0; k < matrix1[0].length; k++) {
                result[i][j] = add(result[i][j],
                        multiply(matrix1[i][k], matrix2[k][j]));
            }
        }
    }
    return result;
}
```


(c) Paul Fodor & Pearson Inc.

```
/** Print matrices, the operator, and their operation result
 */
public static void printResult(Number[][] m1, Number[][] m2,
           Number[][] m3, char op) {
    for (int i = 0; i < m1.length; i++) {
        for (int j = 0; j < m1[0].length; j++) {
            System.out.print(" " + m1[i][j]);
        }
        if (i == m1.length / 2) {
            System.out.print("  " + op + "  ");
        } else {
            System.out.print("     ");
        }
        for (int j = 0; j < m2.length; j++) {
            System.out.print(" " + m2[i][j]);
        }
        if (i == m1.length / 2) {
            System.out.print("  =  ");
        } else {
            System.out.print("     ");
        }
        for (int j = 0; j < m3.length; j++) {
            System.out.print(m3[i][j] + " ");
        }
        System.out.println();
    }
}
```


(c) Paul Fodor & Pearson Inc.

```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;
    }
```


(c) Paul Fodor & Pearson Inc.

```java
public static void main(String[] args) {
    // Create Integer arrays m1, m2
    Integer[][] m1 = new Integer[][]{{1, 2, 3},{4, 5, 6},{1, 1, 1}};
    Integer[][] m2 = new Integer[][]{{1, 1, 1},{2, 2, 2},{0, 0, 0}};

    // Create an instance of IntegerMatrix
    IntegerMatrix integerMatrix = new IntegerMatrix();

    System.out.println("\nm1 + m2 is ");
    GenericMatrix.printResult(
            m1, m2, integerMatrix.addMatrix(m1, m2), '+');

    System.out.println("\nm1 * m2 is ");
    GenericMatrix.printResult(
            m1, m2, integerMatrix.multiplyMatrix(m1, m2), '*');
}
```

Output:

```
m1 + m2 is
 1 2 3     1 1 1    2 3 4
 4 5 6  +  2 2 2 =  6 7 8
 1 1 1     0 0 0    1 1 1

m1 * m2 is
 1 2 3     1 1 1    5 5 5
 4 5 6  *  2 2 2 =  14 14 14
 1 1 1     0 0 0    3 3 3
```


(c) Paul Fodor & Pearson Inc.

```java
public class Rational extends Number implements Comparable<Rational> {
    // Data fields for numerator and denominator

    private long numerator = 0;
    private long denominator = 1;

    /**
     * Construct a rational with specified numerator and denominator
     */
    public Rational(long numerator, long denominator) {
        long gcd = gcd(numerator, denominator);
        this.numerator = ((denominator > 0) ? 1 : -1) * numerator / gcd;
        this.denominator = Math.abs(denominator) / gcd;
    }
    private static long gcd(long n, long d) {
        long n1 = Math.abs(n);
        long n2 = Math.abs(d);
        int gcd = 1;

        for (int k = 1; k <= n1 && k <= n2; k++) {
            if (n1 % k == 0 && n2 % k == 0) {
                gcd = k;
            }
        }

        return gcd;
    }
```


(c) Paul Fodor & Pearson Inc.

```java
public class Rational extends Number implements Comparable<Rational> {
    // Data fields for numerator and denominator

    private long numerator = 0;
    private long denominator = 1;

    /**
     * Construct a rational with specified numerator and denominator
     */
    public Rational(long numerator, long denominator) {
        long gcd = gcd(numerator, denominator);
        this.numerator = ((denominator > 0) ? 1 : -1) * numerator / gcd;
        this.denominator = Math.abs(denominator) / gcd;
    }
    private static long gcd(long n, long d) {
        long n1 = Math.abs(n);
        long n2 = Math.abs(d);
        int gcd = 1;

        for (int k = 1; k <= n1 && k <= n2; k++) {
            if (n1 % k == 0 && n2 % k == 0) {
                gcd = k;
            }
        }

        return gcd;
    }
```


(c) Paul Fodor & Pearson Inc.

```
/**
 * Add a rational number to this rational
 */
public Rational add(Rational secondRational) {
    long n = numerator * secondRational.getDenominator()
            + denominator * secondRational.getNumerator();
    long d = denominator * secondRational.getDenominator();
    return new Rational(n, d);
}
/**
 * Multiply a rational number to this rational
 */
public Rational multiply(Rational secondRational) {
    long n = numerator * secondRational.getNumerator();
    long d = denominator * secondRational.getDenominator();
    return new Rational(n, d);
}
@Override
public String toString() {
    if (denominator == 1) {
        return numerator + "";
    } else {
        return numerator + "/" + denominator;
    }
}
```


(c) Paul Fodor & Pearson Inc.

```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);
    }
```


(c) Paul Fodor & Pearson Inc.

```
public static void main(String[] args) {
    // Create two Rational arrays m1 and m2
    Rational[][] m1 = new Rational[3][3];
    Rational[][] m2 = new Rational[3][3];
    for (int i = 0; i < m1.length; i++) {
        for (int j = 0; j < m1[0].length; j++) {
            m1[i][j] = new Rational(i + 1, j + 5);
            m2[i][j] = new Rational(i + 1, j + 6);
        }
    }

    // Create an instance of RationalMatrix
    RationalMatrix rationalMatrix = new RationalMatrix();

    System.out.println("\nm1 + m2 is ");
    GenericMatrix.printResult(
            m1, m2, rationalMatrix.addMatrix(m1, m2), '+');

    System.out.println("\nm1 * m2 is ");
    GenericMatrix.printResult(
            m1, m2, rationalMatrix.multiplyMatrix(m1, m2), '*');
}
```


(c) Paul Fodor & Pearson Inc.

Output:

m1 + m2 is

```
 1/5 1/6 1/7       1/6 1/7 1/8      11/30 13/42 15/56
 2/5 1/3 2/7   +    1/3 2/7 1/4  =   11/15 13/21 15/28
 3/5 1/2 3/7       1/2 3/7 3/8      11/10 13/14 45/56
```

m1 * m2 is

```
 1/5 1/6 1/7       1/6 1/7 1/8      101/630 101/735 101/840
 2/5 1/3 2/7   *    1/3 2/7 1/4  =   101/315 202/735 101/420
 3/5 1/2 3/7       1/2 3/7 3/8      101/210 101/245 101/280
```


(c) Paul Fodor & Pearson Inc.