Type Erasure

…

What is Type Erasure?

The way for the Java compiler to

- generate byte code which implements our Generic Types & Generic Methods

Type erasure ensures that…

- no new classes are created for parameterized types
- consequently, generics incur no runtime overhead

What does it entail to apply Type Erasure?

To implement generics, the Java compiler applies type erasure to:

1. Replace all type parameters in generic types with their bounds or Object if the type parameters are unbounded. The produced bytecode, therefore, contains only ordinary classes, interfaces, and methods

2. Insert type casts if necessary to preserve type safety

3. Generate bridge methods to preserve polymorphism in extended generic types

Type Erasure of Generic Types

Let's start w/ a class using a non-bounded type parameter

```
public class Node<T> {

    private T data;
    private Node<T> next;

    public Node(T data, Node<T> next) }
        this.data = data;
        this.next = next;
    }

    public T getData() { return data; }
    // ...
}
```

Type Erasure of Generic Types

What happens during Type Erasure?

T is unbounded
→ replace w/ “Object”

```
public class Node {

    private Object data;
    private Node next;

    public Node(Object data, Node next) {
        this.data = data;
        this.next = next;
    }

    public Object getData() { return data; }
    // ...
}
```

Type Erasure of Generic Types

What if our class uses a bounded type parameter?

```
public class Node<T extends Comparable<T>> {

    private T data;
    private Node<T> next;

    public Node(T data, Node<T> next) {
        this.data = data;
        this.next = next;
    }

    public T getData() { return data; }
    // ...
}
```

Type Erasure of Generic Types

What happens during Type Erasure?

Java replaces T by leftmost / 1st bound class: comparable

```
public class Node {

    private Comparable data;
    private Node next;

    public Node(Comparable data, Node next) {
        this.data = data;
        this.next = next;
    }

    public Comparable getData() { return data; }
    // ...
}
```

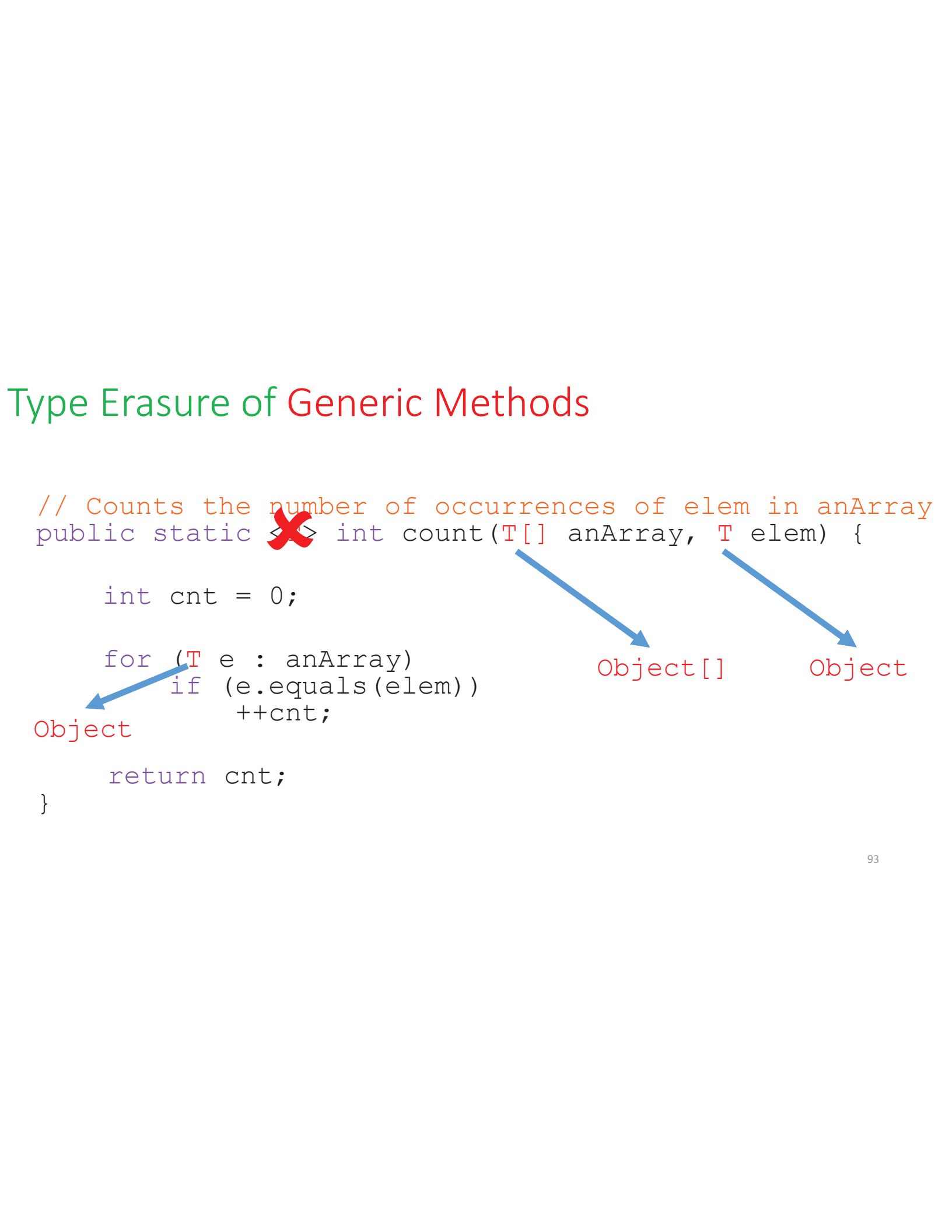
Type Erasure of Generic Methods

```
// Counts the number of occurrences of elem in anArray
public static <T> int count(T[] anArray, T elem) {

    int cnt = 0;

    for (T e : anArray)
        if (e.equals(elem))
            ++cnt;

    return cnt;
}
```

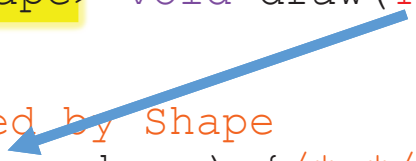


Example

```
// Let's consider a few classes
class Shape                         {/*…*/}
class Circle extends Shape          {/*…*/}
class Rectangle extends Shape       {/*…*/}

// Define generic method to draw different shapes
public static <T extends Shape> void draw(T shape) {/*…*/}

// Type Erasure → T replaced by Shape
public static void draw(Shape shape) {/*…*/}
```



Type Erasure

Effect of Type Erasure & Bridge Methods

Let's consider these 2 classes…

```
public class Node<T> {

    public T data;

    public Node(T data) { this.data = data; }

    public void setData(T data) {
        System.out.println("Node.setData");
        this.data = data;
    }
}
```

```
public class MyNode extends Node<Integer> {
    public MyNode(Integer data) { super(data); }

    public void setData(Integer data) {
        System.out.println("MyNode.setData");
        super.setData(data);
    }
}
```

Now, let's consider the following code…

Here comes the Type Erasure…

```
MyNode mn = new MyNode(5);

Node n = mn;
// n is of raw type Node
// → compiler throws unchecked warning

n.setData("Hello");

Integer x = mn.data;
// → ClassCastException is thrown
```

```
Node n = (MyNode)mn;
```

Compiler adds cast since mn is of class MyNode & MyNode extends an erased type

```
Integer x = (String)mn.data;
```

Erased field is of type Object

Explanation – what happens when running this code?

```
MyNode mn = new MyNode(5);

Node n = (MyNode)mn;
n.setData("Hello");

Integer x = (String)mn.data;
```

setData(Object) is executed on MyNode object

MyNode class inherited setData(Object) from Node after it was type-erased

Inside setData(Object) from Node class…

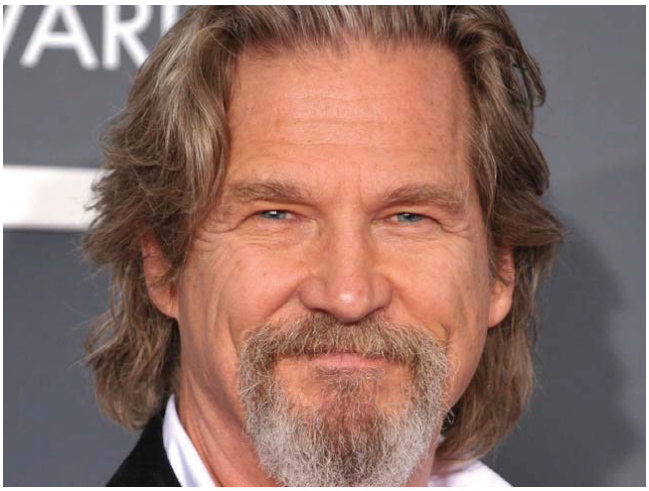
…data field is assigned the String reference

Use mn to access the object's data field…

Result expected to be integer since mn is MyNode which is Node<Integer>

assign String to Integer?
→ ClassCastException from cast inserted by Java compiler

Introducing… Bridge Methods



After Type Erasure, our 2 previous classes become…

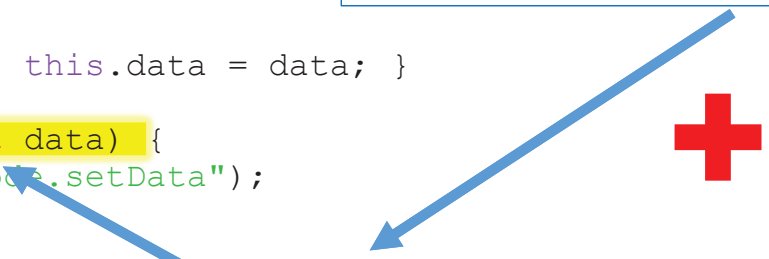
```
public class Node {

    public Object data;

    public Node(Object data) { this.data = data; }

    public void setData(Object data) {
        System.out.println("Node.setData");
        this.data = data;
    }
}
```

```
// Bridge method generated
public void setData(Object data) {
        setData((Integer) data);
}
```



```
public class MyNode extends Node {

    public MyNode(Integer data) { super(data); }

    public void setData(Integer data) {
        System.out.println("MyNode.setData");
        super.setData(data);
    }
}
```

Methods signatures no longer match…

→ This is not overriding anymore!