Documentation

The Java™ Tutorials

Trail: Learning the Java Language
Lesson: Generics (Updated)
Section: Type Erasure

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Effects of Type Erasure and Bridge Methods

Sometimes type erasure causes a situation that you may not have anticipated. The following example shows how this can occur. The example (described in Bridge Methods) shows how a compiler sometimes creates a synthetic method, called a bridge method, as part of the type erasure process.

Given the following two 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);
    }
}
```

Consider the following code:

```
MyNode mn = new MyNode(5);
Node n = mn;            // A raw type - compiler throws an unchecked warning
n.setData("Hello");
Integer x = mn.data;    // Causes a ClassCastException to be thrown.
```

After type erasure, this code becomes:

```
MyNode mn = new MyNode(5);
Node n = (MyNode)mn;         // A raw type - compiler throws an unchecked warning
n.setData("Hello");
Integer x = (String)mn.data; // Causes a ClassCastException to be thrown.
```

Here is what happens as the code is executed:

- `n.setData("Hello");` causes the method `setData(Object)` to be executed on the object of class `MyNode`. (The `MyNode` class inherited `setData(Object)` from `Node`.)
- In the body of `setData(Object)`, the data field of the object referenced by `n` is assigned to a `String`.

- The data field of that same object, referenced via `mn`, can be accessed and is expected to be an integer (since `mn` is a `MyNode` which is a `Node<Integer>`.
- Trying to assign a `String` to an `Integer` causes a `ClassCastException` from a cast inserted at the assignment by a Java compiler.

Bridge Methods

When compiling a class or interface that extends a parameterized class or implements a parameterized interface, the compiler may need to create a synthetic method, called a *bridge method*, as part of the type erasure process. You normally don't need to worry about bridge methods, but you might be puzzled if one appears in a stack trace.

After type erasure, the `Node` and `MyNode` 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;
    }
}

public class MyNode extends Node {

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

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

After type erasure, the method signatures do not match. The `Node` method becomes `setData(Object)` and the `MyNode` method becomes `setData(Integer)`. Therefore, the `MyNode` `setData` method does not override the `Node` `setData` method.

To solve this problem and preserve the polymorphism of generic types after type erasure, a Java compiler generates a bridge method to ensure that subtyping works as expected. For the `MyNode` class, the compiler generates the following bridge method for `setData`:

```
class MyNode extends Node {

    // Bridge method generated by the compiler
    //
    public void setData(Object data) {
        setData((Integer) data);
    }

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

    // ...
}
```

As you can see, the bridge method, which has the same method signature as the `Node` class's `setData` method after type erasure, delegates to the original `setData` method.

Previous page: Erasure of Generic Methods
Next page: Non-Reifiable Types