COMP249 Tutorial 8: Generics and Lambdas

Ella Noyes

August 4, 2024

1 Generics

Generics make code more generic (i.e., generalized or inclusive, as opposed to type-specific). Using generics means that code can operate on objects of different types. Generics make for more flexible and reusable structures and algorithms.

1.1 Syntax

A generic class is declared in Java using the following syntax:

```
public class ClassName<T> {
    ...
}
```

The angled brackets are the essential component in this. T is the "type parameter", it will be used to refer to objects of that generic type elsewhere in the class. You might, for example, have a data member of type T with a setter and a getter.

```
public class ClassName<T> {
    private T myObject;

    ...


    public T getMyData() {
        return myObject;
    }

    public void setMyData(T object) {
        this.myObject = objectl
    }
}
```

You can also write generic methods:

```
public <T> returnType methodName(){
    ...
}
```

Let's look at some concrete examples in code.

1.2 Last Remarks on Generics

You can restrict the types that generic methods/classes may use by restricting the type parameter using the **extends** keyword.

```
public <T extends MyClass> returnType methodName(){
    ...
}
```

T may only be be MyClass or one of its subclasses.

Type erasure: From ChatGPT: Java uses type erasure to implement generics, meaning generic type information is not available at runtime. This allows for backward compatibility with older Java versions but limits some runtime type operations.

2 Lambda Expressions

A lambda expression represents an instance of a functional interface. A functional interface is an interface with only one abstract method, lambda expressions exploit this fact, allowing you to implement that abstract method in a compact expression.
The syntax of a lambda expression is simple:

```
(parameters) -> expression
```

The parameters represent those declared in the abstract method, and the expression implements that abstract method

Let's look at some code examples

3 Programming Problems

3.1 Recursion Problems (From the Tutorial Slides on Moodle)

1. Write a recursive function named reverse (String s) that returns the reverse of a String
   Example:
   Input: Wonderful Day
   Output: yaD lufrednoW

2. Write a recursive method called convertToBinary(int n) that takes an integer and returns the binary representation of the number as a String.
   Example:
   Input: 14
   Output: 1110

3.2 Lambdas and Generics Practice: Integer Operations

My problem, formulated by ChatGPT (Note: use for-loops for isPrime and factorial, not recursion!).

Problem Statement

You are tasked with implementing and using lambda expressions to perform various operations on integers. Define a custom functional interface and write lambda expressions to perform the following tasks:

1. **Check if a Number is Prime:** Write a lambda expression that takes an integer and returns a boolean value indicating whether the number is a prime number. A prime number is a positive integer greater than 1 that has no positive integer divisors other than 1 and itself.

2. **Convert Number to Binary:** Write a lambda expression that takes an integer and returns its binary representation as a string. For example, the binary representation of the decimal number 7 is "111".

3. **Calculate Factorial:** Write a lambda expression that takes an integer and returns its factorial. The factorial of a non-negative integer n is the product of all positive integers less than or equal to n. For example, the factorial of 5 (denoted as 5!) is $5 \times 4 \times 3 \times 2 \times 1 = 120$.

Requirements

1. Define a custom functional interface `IntOperation` using generics to handle different types of results.

2. Implement the following lambda expressions using the `IntOperation` interface:

   - `isPrime`: Checks if a number is prime.
   - `toBinary`: Converts a number to its binary representation.
   - `factorial`: Calculates the factorial of a number.

3. Create a main class to test these lambda expressions with various integer inputs.

Example

Given the integer $n = 7$:

- `isPrime.apply(7)` should return `true`.
- `toBinary.apply(7)` should return `"111"`.
- `factorial.apply(7)` should return `5040`.