Inner Classes, Lambdas and Streams

By Warren Mansur; edited by Eric Braude

BOSTON UNIVERSITY
METROPOLITAN COLLEGE

Inner Classes, Lambdas, and Streams

1. **Inner Classes**
2. Lambdas
3. Streams
4. Object I/O

Inner Classes

- Class nested within another class or method
- Useful for grouping classes
- --or where class only needed inside another class

Nested Inner Class

```
public class OuterClass {
  private int outerX = 5;

  public class NestedInnerClass {
    private int innerY = outerX + 5;

    public int GetInnerY() {
      return innerY;
    }
  }
}
public class UseBasicInnerClass {
  public static void main(String[] args) {
    OuterClass outerClass = new OuterClass();
    OuterClass.NestedInnerClass innerClass =
       outerClass.new NestedInnerClass();
    System.out.println(innerClass.GetInnerY());
  }
}
```

Output

```
10
```

Method-local Inner Class

```
public class MethodLocalInnerClass {
  private static int classMember = 5;
  public static void main(String[] args) {
    int methodVariable = 10;
    class InnerClass {
      public int getInnerValue() {
        return classMember + methodVariable;
      }
    }
    InnerClass innerClass = new InnerClass();
    System.out.println(innerClass.getInnerValue());
  }
}
```

Output

```
15
```

Static Inner Classes

- Not tied to an object
- Can be instantiated independently of other objects
- Can access private static variables from the outer class, but not member variables

Static Inner Class

```
public class BasicOuterClass {
  private static int x = 5;
  private int y = 17;

  public static class StaticInnerClass {
    private int innerVar = x + 10;
    //Cannot access y because it's not static.

    public int getInnerVar() {
      return innerVar;
    }
  }
}

public class StaticInnerClassDemo {
  public static void main(String[] args) {
    BasicOuterClass.StaticInnerClass staticInner =
              new BasicOuterClass.StaticInnerClass();
    System.out.println(staticInner.getInnerVar());
  }
}
```

Output

15

```
public class Show {
  protected int showMemberVar = 5;

  public void ShowIt() {
    System.out.println("In ShowIt() method of Show");
  }
}
public class AnonymousClassDemo {
  public static void main(String[] args) {
    int localVar = 25;
    Show show = new Show() {
      public void ShowIt() {
        super.ShowIt();
        System.out.println("showMemberVar=" +
             showMemberVar + " and localVar=" + localVar);
      }
    };
    show.ShowIt();
  }
}
```

Anonymous Inner Class

Output

```
In ShowIt() method of Show
showMemberVar=5 and localVar=25
```

BOSTON UNIVERSITY METROPOLITAN COLLEGE

Inner Classes, Lambdas, and Streams

1. Inner Classes
2. **Lambdas**
3. Streams
4. Object I/O

Lambdas

- Lambda calculus classic computer science
- Functions represented anonymously
- e.g., *add*(x, y) = x + y represented as (x,y) -> x+y
- Java uses classic syntax--simple functions defined on the fly
- In Java, a lambda is tied to a functional interface – an interface consisting of a single abstract method
- Thus, lambdas can be passed around as arguments, like objects

Lambda Usage

```
public class BasicLambda {
  static interface TwoArgOperation {
    int operation(int arg1, int arg2);
  }

  public static void main(String[] args) {
    TwoArgOperation subtraction = (x, y) -> x - y;
    TwoArgOperation addition = (x, y) -> x + y;
    TwoArgOperation addTwice = (x, y) -> {
      int tmp = x + y;
      return tmp + tmp;
    };

    System.out.println("10 - 7 = " + subtraction.operation(10, 7));
    System.out.println("10 + 7 = " + addition.operation(10, 7));
    System.out.println("10 + 7 added twice = " + addTwice.operation(10, 7));
  }
}
```

Output

```
10 - 7 = 3
10 + 7 = 17
10 + 7 added twice = 34
```

BOSTON UNIVERSITY
METROPOLITAN COLLEGE

Inner Classes, Lambdas, and Streams

1. Inner Classes
2. Lambdas
3. **Streams**
4. Object I/O

Streams *1 of 2*

- A way to concisely process sequences of objects
- Transforms Collections, Arrays, and I/O channels
- Don’t modify the underlying data
- Three operations – creation, intermediate operations, and terminal operations.

Streams *2 of 2*

- Can be evaluated in parallel
- Pipelined—when one operation has finished working on part, next operation can start working on it
  - without waiting for entire collection to be processed
- Mostly declarative
  - e.g., similar operations to SQL

Basic Stream Example

```
public class BasicStreamDemo {
  public static void main(String[] args) {
    Stream<Integer> intStream = Stream.of(1, 2, 3, 4);
    intStream.filter(i -> i > 2).map(i -> i+10).forEach(i -> System.out.println(i));
  }
}
```

```
Output
13
14
```

Creating Streams

- Stream.of() (array or variable arguments)
- stream() in the *Collection* interface
  - e.g., Lists and Sets
- Arrays.stream() static method
- Stream.builder() gives Stream.Builder, which has methods to build a stream
- …

Intermediate Operations

- filter()
  - creates new stream, eliminating objects not matching condition
- map()
  - creates a new stream with one or more operations applied to the object (a stream with new values)
- sorted()
  - either natural order or with a defined comparator
- distinct()
  - provides a distinct list of objects

Terminal Operations *1 of 2*

- forEach()
  - performs an operation on each object, e.g., printing it
- allMatch()
  - returns true or false depending upon whether *all* objects meet condition
- anyMatch()
  - returns true or false depending upon whether *any* object meets the condition

Terminal Operations *1 of 2*

- **collect()**
  - **collects the objects into a *Collection* such as a list or set**
- **reduce()**
  - **reduces the set of objects to a single object**

Tips on Streams

- Use when collections of objects must be manipulated
  - much more efficient and concise than writing code to perform every step
- Beware modifying the collection in the stream
  - operations can result in dangerous or inconsistent behavior

BOSTON UNIVERSITY
METROPOLITAN COLLEGE

Inner Classes, Lambdas, and Streams

1. Inner Classes
2. Lambdas
3. Streams
4. **Object I/O**

What is Object I/O?

- Write objects (instances of classes) whole
- Read
- Use ObjectOutputStream

Writing an Object

```
try {
    // Create a new file and write the request object
    FileOutputStream fileOut =
        new FileOutputStream(FILE_OF_REQUEST_OBJECTS);
    ObjectOutputStream out = new ObjectOutputStream(fileOut);
    out.writeObject(aRequest);
    out.close();
    fileOut.close();
}
} catch (IOException i) …
```

```
try {
    if (Files.exists(Paths.get(FILE_OF_REQUEST_OBJECTS))) {
        // Append the request object to the existing file
        FileOutputStream fileOut = new FileOutputStream
                (FILE_OF_REQUEST_OBJECTS, true);
        ObjectOutputStream out = new ObjectOutputStream(fileOut) {
                @Override
                protected void writeStreamHeader() throws IOException {
                        reset(); // Reset the stream header to avoid conflicts
                }
        };
        out.writeObject(aRequest);
        System.out.println("Request object " + aRequest.toString() + " stored");
        out.close();
        fileOut.close();
    } else {
    // Create a new file and write the request object
        FileOutputStream fileOut = new FileOutputStream(FILE_OF_REQUEST_OBJECTS);
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        out.writeObject(aRequest);
        System.out.println("Request object " + aRequest.toString() + " stored");
        out.close();
        fileOut.close();
} catch (IOException i) {
```

Append mode

A block data record is composed of a header and data

Appending an Object on File

Reading an Object from a File

```
try {
    // Read the request objects from the file
    FileInputStream fileIn = new FileInputStream(FILE_OF_REQUEST_OBJECTS);
    ObjectInputStream in = new ObjectInputStream(fileIn);

    while (true) {
        try {
            Request<Action> request = (Request<Action>) in.readObject();
            System.out.println
                ("Read request: " + request.getAction().toString());
        } catch (EOFException e) { // end of file
        break;
        }
    }
    in.close();
    fileIn.close();
} catch …
```