Core Java

Day16 Agenda

- Stream programming
  - Stream operations
  - Optional
  - Primitive streams
- Method references
- enum
- Reflection

Stream Programming

Operations

- Refer yesterday's notes.

Collect Stream result

- Collecting stream result is terminal operation.
- Object[] toArrray()
- R collect(Collector)
  - Collectors.toList(), Collectors.toSet(), Collectors.toCollection(), Collectors.joining()
  - Collectors.toMap(key, value)

Stream of primitive types

- Efficient in terms of storage and processing. No auto-boxing and unboxing is done.
- IntStream class
  - IntStream.of() or IntStream.range() or IntStream.rangeClosed() or Random.ints()
  - sum(), min(), max(), average(), summaryStatistics(),
  - OptionalInt reduce().

Method references

- If lambda expression involves single method call, it can be shortened by using method reference.
- Method references are converted into instances of functional interfaces.
- Method reference can be used for class static method, class non-static method, object non-static method or constructor.

Examples

- Class static method: Integer::sum [(a,b) -> Integer.sum(a,b)]
  - Both lambda param passed to static function explicitly
- Class non-static method: String::compareTo [(a,b) -> a.compareTo(b)]

    - First lambda param become implicit param (this) of the function and second is passed explicitly (as arguments).
- Object non-static method: System.out::println [x -> System.out.println(x)]
    - Lambda param is passed to function explicitly.
- Constructor: Date::new [() -> new Date()]
    - Lambda param is passed to constructor explicitly.

enum

- "enum" keyword is added in Java 5.0.
- Used to make constants to make code more readable.
- Typical switch case

```java
int choice;
// ...
switch(choice) {
    case 1: // addition
        c = a + b;
        break;
    case 2: // subtraction
        c = a - b;
        break;
    // ...
}
```

- The switch constants can be made more readable using Java enums.

```java
enum ArithmeticOperations {
    ADDITION, SUBTRACTION, MULIPLICATION, DIVISION;
}

ArithmeticOperations choice = ArithmeticOperations.ADDITION;
// ...
switch(choice) {
    case ADDITION:
        c = a + b;
        break;
    case SUBTRACTION:
        c = a - b;
        break;
    // ...
}
```

- In java, enums cannot be declared locally (within a method).
- The declared enum is converted into enum class.

```java
// user-defined enum
enum ArithmeticOperations {
    ADDITION, SUBTRACTION, MULIPLICATION, DIVISION;
}
```

```java
// generated enum code
final class ArithmeticOperations extends Enum {
    public static ArithmeticOperations[] values() {
        return (ArithmeticOperations[])$VALUES.clone();
    }
    public static ArithmeticOperations valueOf(String s) {
        return (ArithmeticOperations)Enum.valueOf(ArithmeticOperations, s);
    }
    private ArithmeticOperations(String name, int ordinal) {
        super(name, ordinal); // invoke sole constructor Enum(String,int);
    }
    public static final ArithmeticOperations ADDITION;
    public static final ArithmeticOperations SUBTRACTION;
    public static final ArithmeticOperations MULIPLICATION;
    public static final ArithmeticOperations DIVISION;
    private static final ArithmeticOperations $VALUES[];
    static {
        ADDITION = new ArithmeticOperations("ADDITION", 0);
        SUBTRACTION = new ArithmeticOperations("SUBTRACTION", 1);
        MULIPLICATION = new ArithmeticOperations("MULIPLICATION", 2);
        DIVISION = new ArithmeticOperations("DIVISION", 3);
        $VALUES = (new ArithmeticOperations[] {
            ADDITION, SUBTRACTION, MULIPLICATION, DIVISION
        });
    }
}
```

- The enum type declared is implicitly inherited from java.lang.Enum class. So it cannot be extended from another class, but enum may implement interfaces.
- The enum constants declared in enum are public static final fields of generated class. Enum objects cannot be created explicitly (as generated constructor is private).
- The generated class will have a values() method that returns array of all constants and valueOf() method to convert String to enum constant.
- The enums constants can be used in switch-case and can also be compared using == operator.
- The java.lang.Enum class has following members:

```java
public abstract class Enum<E> implements java.lang.Comparable<E>,
java.io.Serializable {
    private final String name;
    private final int ordinal;

    protected Enum(String,int); // sole constructor - can be called from
user-defined enum class only
    public final String name(); // name of enum const
    public final int ordinal(); // position of enum const (0-based)

    public String toString(); // returns name of const
    public final int compareTo(E);// compares with another enum of same type
on basis of ordinal number
    public static <T> T valueOf(Class<T>, String);
    // ...
}
```

- The enum may have fields and methods.

```
enum Element {
    H(1, "Hydrogen"),
    HE(2, "Helium"),
    LI(3, "Lithium");

    public final int num;
    public final String label;

    private Element(int num, String label) {
        this.num = num;
        this.label = label;
    }
}
```

Reflection

- .class = Byte-code + Meta-data + Constant pool + ...
- When class is loaded into JVM all the metadata is stored in the object of java.lang.Class (heap area).
- This metadata includes class name, super class, super interfaces, fields (field name, field type, access modifier, flags), methods (method name, method return type, access modifier, flags, method arguments, ...), constructors (access modifier, flags, ctor arguments, ...), annotations (on class, fields, methods, ...).

Reflection applications

- Inspect the metadata (like javap)
- Build IDE/tools (Intellisense)
- Dynamically creating objects and invoking methods
- Access the private members of the class

Get the java.lang.Class object

- way 1: When you have class-name as a String (taken from user or in properties file)

```
Class<?> c = Class.forName(className);
```

- way 2: When the class is in project/classpath.

```
Class<?> c = ClassName.class;
```

- way 3: When you have object of the class.

```
Class<?> c = obj.getClass();
```

Access metadata in java.lang.Class

- Name of the class

```
String name = c.getName();
```

- Super class of the class

```
Class<?> supcls = c.getSuperclass();
```

- Super interfaces of the class

```
Class<?> supintf[] = c.getInterfaces();
```

- Fields of the class

```
Field[] fields = c.getFields(); // all fields accessible (of class & its super class)
```

```
Field[] fields = c.getDeclaredFields(); // all fields in the class
```

- Methods of the class

```
Method[] methods = c.getMethods(); // all methods accessible (of class & its super class)
```

```
Method[] methods = c.getDeclaredMethods(); // all methods in the class
```

- Constructors of the class

  ```
  Constructor[] ctors = c.getConstructors(); // all ctors accessible (of class & its super class)
  ```

  ```
  Constructor[] ctors = c.getDeclaredConstructor(); // all ctors in the class
  ```

Reflection Tutorial

- https://youtu.be/lAoNJ_7LD44
- https://youtu.be/UVWdtk5ibK8