Power Programming in Java

Session: 14

New and Advanced Features of Java

Objectives

- List the enhancements in switch-case
- Explain new FileSystems methods
- Explain the notPredicate method
- Describe Records and TextBlocks
- Explain Flow, StackWalking, and HTTP Client APIs
- Describe accounting style currency format support
- Explain CompactNumberFormat class

Enhancements in switch [1-12]

`switch`

- Is commonly used in Java
- Follows the design of C and C++

Old switch works well as long as `break` statements are included, because of missing break statement the chance of error in code was more.

Enhancements in switch [2-12]

- Supports multiple values per `case`
- Uses `yield` to return a value
- Serves as an expression
- Is necessary to return a value or an exception
- Uses arrows
- Has changed scope

Enhancements in switch [3-12]

- Allows to specify multiple values per `case`.
- Delimits each values by commas.

Code Snippet

```
import java.util.Scanner;
public class EnhancedSwitchDemo {
    public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter Product ID to check the Product
        label:");
    int prodID = sc.nextInt();
    switch (prodID) {
          case 101, 102, 103 :
          System.out.println("You have selected a smartwatch!");
          break;
          case 104, 105:
          System.out.println("You have selected a smartphone!");
          break;
          }
    }
}
```

Enhancements in switch [4-12]

`yield:`

Is similar to `return` statement, but used exclusively with `switch` statement

Is used to return specific value from a `switch` branch

Terminates the switch expression, without the `break` statement

Enhancements in switch [5-12]

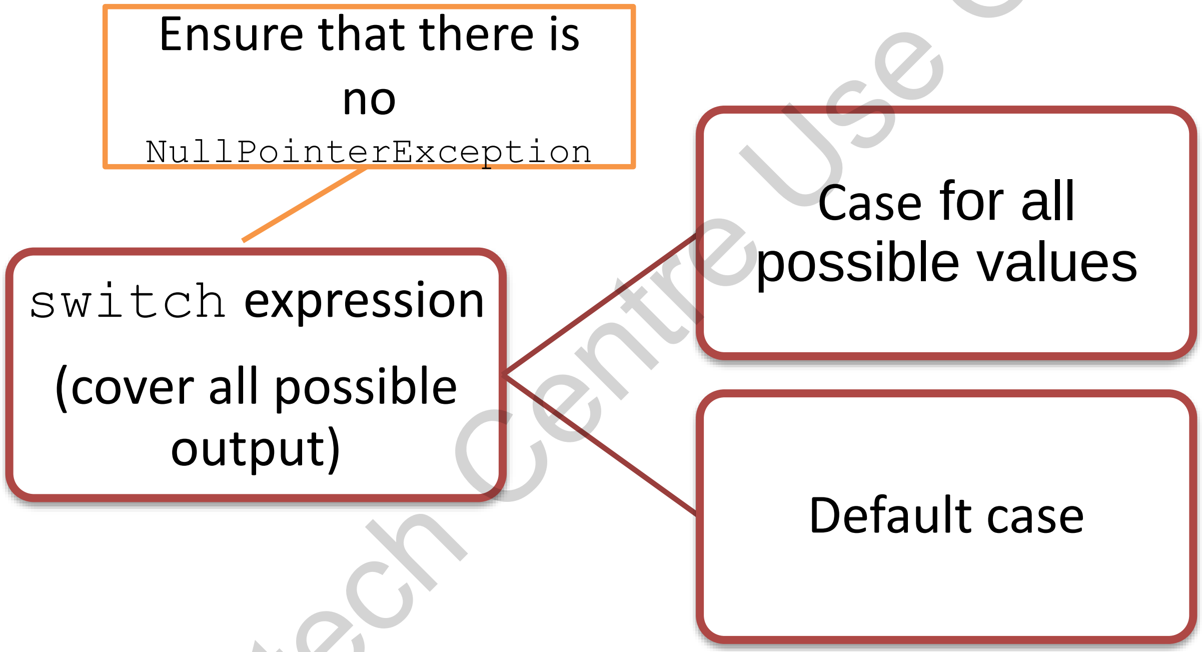
- Serves as a expression too.

Code Snippet

```
import java.util.Scanner;
public class EnhancedSwitchDemo3 {
public static void main(String[]args) {
     Scanner sc = new Scanner(System.in);
     System.out.print("Enter a Code to check State
        stats:");
     int ivar = sc.nextInt();
     //Retrieve the result of a switch expression
     //and assign it to a variable.
     String numberYieldColon = switch(ivar){
         case 0: yield "Texas";
         case 1: yield "California";
         case 2: {
         String colResult = "Exclusively";
         colResult = colResult + "Seattle";
         yield colResult;
         }
         case 3: yield "finally, Chicago";
         default: yield "NA";
         };
         //switch ends with semicolon
         System.out.println("Leading State is
         "+numberYieldColon);
     }
}
```

Enhancements in switch [6-12]

- Is necessary to return a value/expression:


Ensure that there is no NullPointerException
switch expression
(cover all possible output)
Case for all possible values
Default case

It is mandatory for `switch` expression to return some value or explicitly throw an exception, irrespective of the input value.

Enhancements in switch [7-12]

Code Snippet

```
import java.util.Scanner;
public class EnhancedSwitchDemo4 {
  public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
    System.out.print("Enter the Product Name to check the Product 
        availability:");
    String prodName= sc.next();
    int res = getResultViaYield(prodName);
    String status = res==1? "Available" : "Not Available";
    System.out.println("The Product is "+ status);
    }
  private static int getResultViaYield(String name) {
   int result = switch (name) {
        case "Bolt", "Nut":
          //if we enter Bolt or Nut, this yields or returns 1
          yield 1;
        case "Rivet", "Screw":
          //if we enter Rivet or Screw, this yields or returns 2
          yield 2;
        case "Nail":
          //if we enter Nail, this yields or returns 3
          yield 3;
```

Enhancements in switch [8-12]

```
         default:
           throw new IllegalArgumentException(name + " is an unknown
           product and not found in catalog.");
         };
     return result;
   }
}
```

Enhancement in switch [9-12]

arrow syntax

Used in each case and default block

Used both as an expression and a statement

If both left and right statement cases match, then statement present on right side of the arrow are executed

Elements that can be present on the right side of the arrow are:

a throw statement

a statement or an expression

a { } block

Enhancement in switch [10-12]

Code Snippet

```
import java.util.Scanner;
public class EnhancedSwitchDemo5 {
    public static void main(String[] args) {
     Scanner sc= new Scanner(System.in);
     System.out.print("Enter Product ID to check the Product
     label: ");
     int prodId = sc.nextInt();
     System.out.println(getResultViaYield(prodId));
    }
    private static String getResultViaYield(int id) {
      String res = switch (id) {
        case 001 -> "This id represents a smart television";
        case 002 -> "This id represents a smartphone";
        case 003,004 -> "This id represents a smart microwave";
        default -> "Sorry, No match found";
      } ;
     return res;
   }
}
```

Enhancement in switch [11-12]

Has changed scope

- To keep individual case branches as separate scope: Provide a {} block
- Avoid variable clashing

Enhancement in switch [12-12]

Code Snippet

```
import java.util.Scanner;
public class EnhancedSwitchDemo6 {
   public static void main(String[] args) {
   Scanner sc= new Scanner(System.in);
     System.out.print("Enter Product ID to check the Product label: ");
     int prodID = sc.nextInt();
     switch (prodID) {
         case 101: {
            // The num variable exists just in this {} block
            int num = 200;
            System.out.println("The value of num is "+num);
            break;
         }
         case 102: {
           // This is ok, {} block has a separate scope
           int num = 300;
           System.out.println("The value of num is "+num);
           break;
         }
         default:
         {
            System.out.println("Oops! No matches");
         }
     }
   }
}
```

New FileSystems Methods [1-3]

Three new methods that have been added to `FileSystems` class:

- `newFileSystem(Path)`
- `newFileSystem(Path, Map<String, ?>)`
- `newFileSystem(Path, Map<String, ?>, ClassLoader)`

Helps to handle file system providers, which consider the contents of a file as a file system.

New FileSystems Methods [2-3]

Code Snippet

```
//This Java Program illustrates use of new methods of FileSystems class
//Importing URI class from java.net package
import java.net.URI;
//Importing required file classes from java.nio package
import java.nio.file.FileSystem;
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
//Importing Map and HashMap classes from java.util package
import java.util.HashMap;
import java.util.Map;

//Main class
public class NewFileSystemDemo {
public static void main(String[] args) {
try {
Map<String, String> env = new HashMap<>();
// in the following line, we are trying to get path of zip file
Path zipPath = Paths.get("ASample.zip");
// Creating URI from zip path received
URI Uri = new URI("jar:file", zipPath.toUri().getPath(), null);
// Create new file system from URI
FileSystem filesystem = FileSystems.newFileSystem(Uri, env);
```

New FileSystems Methods [3-3]

```
// Display message to inform user
System.out.println("Hurray, you have created File System successfully.");
// Here, we check if file system is open or not, using isOpen()
// method
if (filesystem.isOpen())
     System.out.println("It seems File system is open");
else
     System.out.println("It seems File system is closed");
}
catch (Exception e) {
     // Print the exception with line number
     e.printStackTrace();
     }
  }
}
```

Predicate Not Method [1-3]

- `Predicate.Not()` static method is used negate an existing predicate.
- `Predicate` interface available in `java.util.function` package.
- Can also create another predicate to create a negate and then, assign it with `not()` method.

Code Snippet

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;
public class PredicateNotDemo {
public static void main(String[] args) {
```

Predicate Not Method [2-3]

```
 List<Integer> sampleList
               = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
 /* Let us create a predicate for negation */
 Predicate<Integer> findEven = i -> i % 2 == 0;
 /* Now, we create a predicate object which is negation of supplied
 predicate*/
 Predicate<Integer> findOdd = Predicate.not(findEven);
 /* start filtering the even number using even predicate */
 List<Integer> evenNumbers
              = sampleList.stream().filter(findEven).collect(
                    Collectors.toList());
/* start filtering the odd number using odd predicate */
List<Integer> oddNumbers
         = sampleList.stream().filter(findOdd).collect(
                    Collectors.toList());
  /* Try to print the Lists for odd or even numbers */
  System.out.println("Here is the list of even numbers
          "+evenNumbers);
  System.out.println("Here is the list of odd numbers "+oddNumbers);
    }
}
```

Predicate Not Method [3-3]

- `Predicate.negate()` method

  Creates the logical negation of the existing predicate and then, returns it.

Code Snippet

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;
public class PredicateNegateDemo {
public static void main(String[] args) {
List<Integer> sampleList = Arrays.asList(2020, 2021, 2022, 2023,
2024, 2025, 2026, 2027, 2028, 2029);
//This is to check whether the predicate is leap or not
Predicate<Integer> isLeap = i -> i % 4 == 0;
Predicate<Integer> isNotLeap = isLeap.negate();
List<Integer> leapList = sampleList.stream()
              .filter(isLeap)
              .collect(Collectors.toList());
List<Integer> notLeapList = sampleList.stream()
              .filter(isNotLeap)
              .collect(Collectors.toList());
//print both the lists
System.out.println("Leap Years are "+ leapList);
System.out.println("Not Leap Years are "+ notLeapList );
}
}
```

Records

Records are a new restricted form of class to declare types such as an `enum`.

Is an easy way of defining an immutable data-holding object.

Cannot separate API from the presentation.

These are considered simple and are transparent holders for data.

Automatically acquire multiple standard members.

Text Blocks [1-2]

Text Blocks provide another way to write String literals in source code

Allows to include literal fragments of HTML, JSON, SQL, and so on

Allows to use newlines along with quotes avoiding escape sequence

Text Blocks [2-2]

Code Snippet

```
public class jsonDemo {
    public static void main(String args[]) {
          String json = "{\"name\":\"Dune\",\"year\":2021, "+
          "\"details\":{\"actors\":25,\"budget\":35,
          \"units\":6}," + "\"tags\":[\"films\",\"epic\"],"
          + "\"rating\":9}";
          System.out.println(json);
          }
}
```

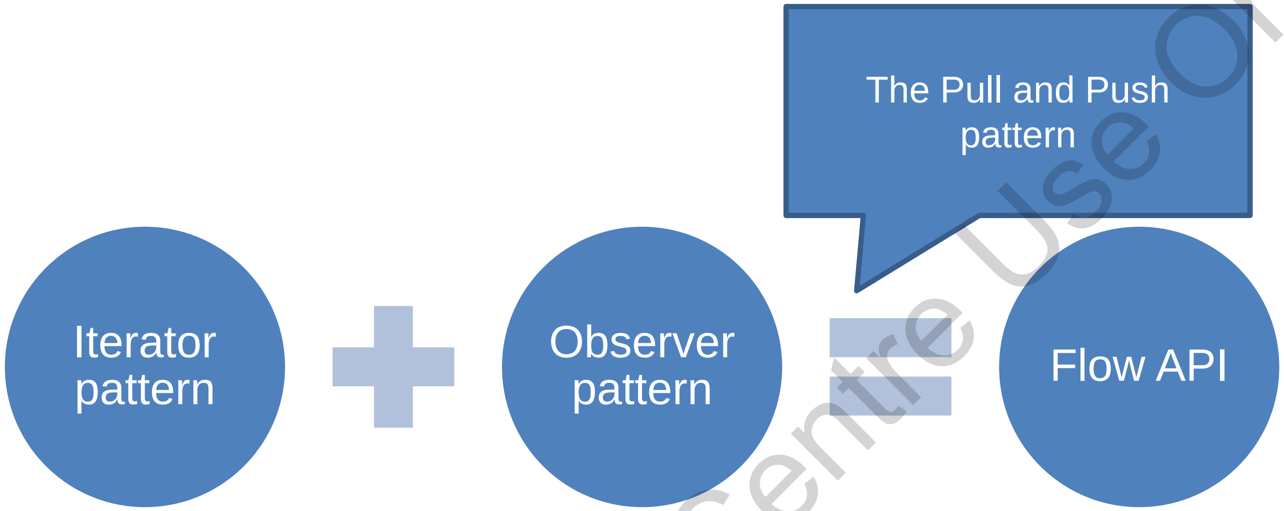
Here, the String literal represents a JSON document embedded in Java code. However, it is accompanied by String concatenation operators (+) and escapes sequences for double quotes ("). The text block feature allows to show multi-line Strings while avoiding escape sequences in general cases. Following Code Snippet shows how you can rewrite the same code using a text block.

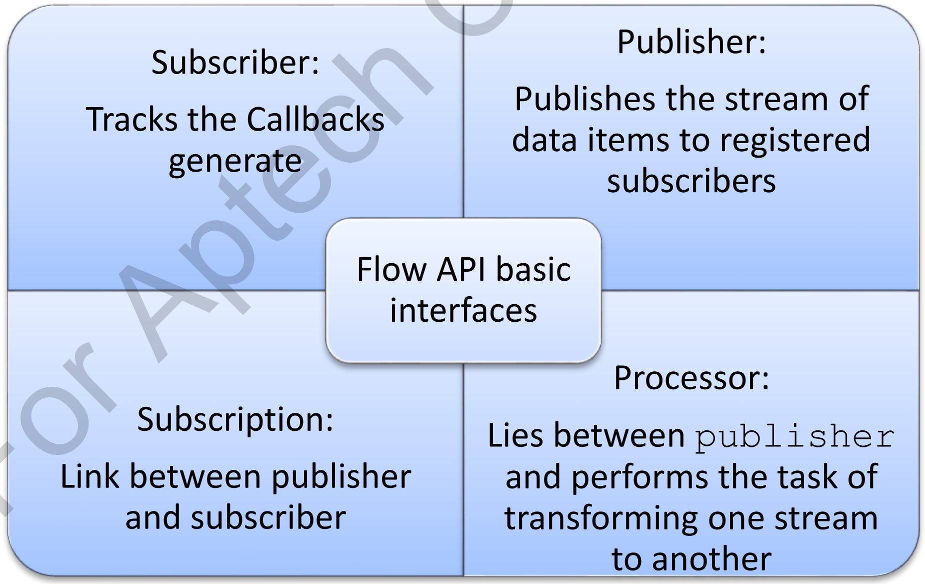
Code Snippet

```
public class jsonDemo {
    public static void main(String args[]){
    String json = """          {
           "name": "Dune", "year": 2021,
           "details": {"actors": 25, "budget (millions": 35, "units": 6}, "tags":
           ["films", "epic"],"rating": 9 }
           """;
           System.out.println(json);
     }
}
```

Flow API [1-3]

- Is the official support for reactive streams specification




Subscriber:
Tracks the Callbacks generate
Publisher:
Publishes the stream of data items to registered subscribers
Flow API basic interfaces
Subscription:
Link between publisher and subscriber
Processor:
Lies between publisher and performs the task of transforming one stream to another

Flow API [2-3]

Code Snippet

```
import java.util.concurrent.Flow;
import java.util.List;
import java.util.concurrent.SubmissionPublisher;
public class SubscriberDemo<T> implements Flow.Subscriber<T> {
        private Flow.Subscription subs;
        @Override
        public void onSubscribe(Flow.Subscription subs) {
            this.subs = subs;
            this.subs.request(1);
        }
        @Override
        public void onNext(T item) {
            System.out.println(item);
            subs.request(1);
        }
        @Override
        public void onError(Throwable throwable) {
            throwable.printStackTrace();
        }
        @Override
        public void onComplete() {
              System.out.println("Control has reached OnComplete method");
         }
       }
```

Flow API [3-3]

On implementing the `Flow.subscriber` class, the methods `onNext()` and `onComplete()` are overridden

Code Snippet

```
//Driver class to demonstrate the working of Flow API
import java.util.concurrent.Flow;
import java.util.List;
import java.util.concurrent.SubmissionPublisher;
public class FlowAPIDemo {
      public static void main(String args[]) {
          List<String> items = List.of("1", "2", "3", "4", "5", "6", "7",
                 "8", "9", "10");
          SubmissionPublisher<String> samplePublisher = new
              SubmissionPublisher<>();
          samplePublisher.subscribe(new SubscriberDemo<>());
          items.forEach(s -> {
                  try {
                       Thread.sleep(1000);
                   } catch (InterruptedException e) {
                      e.printStackTrace();
                   }
                   samplePublisher.submit(s);
                });
                samplePublisher.close();
             }
 }
```

HTTP Client API [1-2]

- Is used to send requests and get back their responses
- Supports HTTP/2 with backward compatibility
- Must be defined in the module using a `module-info.javafile` that indicates required modules to run the application
- Offers both synchronous and asynchronous request mechanisms
- Can be found in `java.net.httppackage`

HTTP Client API [2-2]

- Three core classes in the API are:

`HttpRequest`

- Is for the request to be sent through the `HttpClient`

`HttpClient`

- Is used as a container for configuring information that is common to multiple requests

`HttpResponse`

- Stands for the result of an `HttpRequestcall`

Accounting Currency Format Support [1-2]

- Helps to display money or currency in a format specific to the particular location or country.

  For example: in US it is $

- This is known as Accounting style.

- In Java 14 onwards, this can be done by calling the method `NumberFormat.getCurrencyInstance(locale)` with the `ucfaccountUnicode` locale extension.

Accounting Currency Format Support [2-2]

Code Snippet

```
import java.text.*;
import java.util.*;
public class CurrencyDemo {
    public static void main(String[] args) {
        DecimalFormat df1 = (DecimalFormat)
        DecimalFormat.getCurrencyInstance(Locale.US);
        System.out.println(df1.format(-9.44)); //-$9.44
        Locale myLocale = new Locale.Builder().setLocale(Locale.US)
             .setExtension(Locale.UNICODE_LOCALE_EXTENSION,
             "cf-account").build();
        DecimalFormat df2 = (DecimalFormat)
              NumberFormat.getCurrencyInstance(myLocale);
        System.out.println(df2.format(-9.44)); //Displays ($9.44)
    }
}
```

CompactNumberFormat Class [1-4]

Is a subclass of `NumberFormat` that formats a decimal number based on patterns in a compact form.

Code Snippet

```
import java.text.NumberFormat;
import java.util.Locale;
public class CompactNumberFormatDemo {
          public static void main(String[] args){
              NumberFormat sampleNoFormat = NumberFormat
                    .getCompactNumberInstance(Locale.US, NumberFormat.
                  Style.LONG);
              System.out.println(sampleNoFormat.format(200) );
              System.out.println(sampleNoFormat.format(2000) );
              System.out.println(sampleNoFormat.format(20000) );
              System.out.println(sampleNoFormat.format(200000) );
              NumberFormat sampleShortFormat = NumberFormat
                                       .getCompactNumberInstance(Locale.US,
                    NumberFormat.Style.SHORT);
              System.out.println(sampleShortFormat.format(200) );
              System.out.println(sampleShortFormat.format(2000) );
              System.out.println(sampleShortFormat.format(20000) );
              System.out.println(sampleShortFormat.format(200000) );
          }
}
```

CompactNumberFormat Class [2-4]

- **Custom CompactNumberFormat instance**

  Used to represent numbers in shorter form using the constructor `CompactNumberFormat(String, DecimalFormatSymbols, String[])`

Code Snippet

```
import java.text.CompactNumberFormat;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.Currency;
import java.util.Locale;
public class CustomCompactDemo {
    @SuppressWarnings("deprecation")
    public static void main(String[] args){
        final String[] cmpctPttrns = {"", "", "", "0k", "00k", "000k",
       "0m", "00m", "000m", "0b", "00b", "000b", "0t", "00t",
       "000t"};
       final DecimalFormat decimalFormat = (DecimalFormat)NumberFormat.
       getNumberInstance(Locale.US);
       final CompactNumberFormat customCompactNoFormat=
       new CompactNumberFormat( decimalFormat.toPattern(),
       decimalFormat.getDecimalFormatSymbols(), cmpctPttrns);
       System.out.println(decimalFormat.toPattern());
     }
 }
```

CompactNumberFormat Class [3-4]

- **Set fractional number**

Minimum fraction part digits that are acceptable in a number Default is set to '0' digits.

Code Snippet

```
import java.text.NumberFormat;
import java.util.Locale;
public class FractionFormatDemo {
public static void main(String[] args) {
     NumberFormat format = NumberFormat.getCompactNumberInstance(Locale.US,
               NumberFormat.Style.SHORT);
     format.setMinimumFractionDigits(3);
     System.out.println(format.format(20000));
     System.out.println(format.format(20012));
     System.out.println(format.format(200201));
     System.out.println(format.format(2222222));
      }
}
```

CompactNumberFormat Class [4-4]

- **Compact number parsing**

Is a process used to parse compact number into a long pattern.

Code Snippet

```
import java.text.NumberFormat;
import java.util.Locale;
public class ParsingFormatDemo {
     public static void main(String[] args) throws Exception
     {
          NumberFormat format = NumberFormat
                    .getCompactNumberInstance(Locale.US, NumberFormat.Style.LONG);
          System.out.println(format.parse("200") );
          System.out.println(format.parse("2 thousand") );
          System.out.println(format.parse("20 thousand") );
          System.out.println(format.parse("200 thousand") );
      }
}
```

Summary

- The `switch` statement has been enhanced in many ways in recent Java versions.
- With the multiple value case option, it is possible to provide many values at the same time in a single case.
- A new keyword yield can be used with switch-case blocks to return values.
- `switch` can now be used as an expression to assign a value to another object.
- `FileSystems` class has three new methods.
- `StackWalking` APIs require the VM to capture a snapshot of the entire stack and return information representing it.
- An `HttpClient` provides configuration information and resource sharing for all requests sent through it.
- `Records` are a new type in Java.
- Version Java 14 onwards includes support for currency number accounting styles.
- `CompactNumberFormat` class enables developers to format numbers into compact values.