Core Java 8 and Development Tools

Lesson 05 : Exploring Basic Java Class Libraries

Capgemini

Lesson Objectives

After completing this lesson, participants will be able to:

- Understand The Object Class and different Wrapper Classes
- Use Type casting
- Work with Scanner, System Class and String Handling
- Understand new Date and Time API
- Best Practices

5.1: The Object Class
The Object Class

Cosmic super class

Ultimate ancestor

- Every class in Java implicitly extends Object

Object type variables can refer to objects of any type:

Example:

```
Object obj = new Emp();
```

5.1: The Object Class
Object Class Methods

Method	Description
`boolean equals(Object)`	Determines whether one object is equal to another
`void finalize()`	Called before an unused object is recycled.
`class getClass()`	Obtains the class of an object at run time.
`int hashCode()`	Return the hashcode associated with the invoking object.
`String toString()`	Returns a string that describes the object

5.2: Wrapper Classes
Wrapper Classes

Correspond to primitive data types in Java

Represent primitive values as objects

Wrapper objects are immutable

Simple Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean
void	Void

5.2: Wrapper Classes
Integer Wrapper Class

Integer class wraps a value of primitive type “int” into an object

This class also provides several methods to convert int to String and vice versa

Important methods of Integer class:

- intValue() : retrieves primitive int value of the Integer object
- compareTo(): compares two Integer Objects
- parseInt(): static method used to convert String value to int
- toString(): retrives as String value from Integer object
- isNaN(): check whether the given values is number or not

Important Constants of Integer class:

- MAX_VALUE: represents largest value of Integer class range
- MIN_VALUE: represent lowest value of Integer class range

```
String strValue = “1234”;
int num = Integer.parseInt(strValue);
```

5.3: Type casting
Casting for Conversion of Data type

Casting operator converts one variable value to another where two variables correspond to two different data types

```
variable1 = (variable1) variable2
```

Here, variable2 is typecast to variable1

Data type can either be a reference type or a primitive one

5.3: Type casting
Casting Between Primitive Types

When one type of data is assigned to another type of variable, *automatic type conversion* takes place if:

- Both types are compatible
- Destination type is larger than the source type
- No explicit casting is needed (widening conversion)

```
int a=5; float b; b=a;
```

If there is a possibility of data loss, explicit cast is needed:

```
int i = (int) (5.6/2/7);
```

5.3: Type casting
Casting Between Reference Types

- One class types involved must be the same class or a subclass of the other class type
- Assignment to different class types is allowed only if a value of the class type is assigned to a variable of its superclass type
- Assignment to a variable of the subclass type needs explicit casting:

```
String StrObj = Obj;
```

- Explicit casting is not needed for the following:

```
String StrObj = new String("Hello");
Object  Obj = StrObj;
```

5.3: Type casting
Casting Between Reference Types (contd..)

Two types of reference variable castings:

- Downcasting:

```
Object Obj = new Object ( );
String StrObj = (String) Obj;
```

- Upcasting:

```
String StrObj = new String("Hello");
Object  Obj = StrObj;
```

5.4: Using Scanner Class
Scanner Class

Prior to Java 1.5 getting input from the console involved multiple steps.

Java 1.5 introduced the *Scanner* class to simplify console input.

Also reads from files and Strings (among other sources).

Used for powerful pattern matching.

Scanner is in the Java.util package; therefore needs to be imported

```
import java.util.Scanner;
```

5.4: Using Scanner Class
Creating Scanner Objects

- Scanner(File source): Constructs a new Scanner that produces values scanned from the specified file.
- Scanner(InputStream source): Constructs a new Scanner that produces values scanned from the specified input stream.
- Scanner(Readable source): Constructs a new Scanner that produces values scanned from the specified source.
- Scanner(String source): Constructs a new Scanner that produces values scanned from the specified string.

5.4: Using Scanner Class
How to use Scanner class?

Scanner class basically parses input from the source into tokens by using delimiters to identify the token boundaries.

The default delimiter is whitespace.

Example:

```
Scanner sc = new Scanner (System.in);
int i = sc.nextInt();
System.out.println("You entered" + i);
```

5.4: Using Scanner Class
Scanner class : nextXXX() Methods

String next()

boolean nextBoolean()

byte nextByte()

double nextDouble()

float nextFloat()

int nextInt()

String nextLine()

long nextLong()

short nextShort()

5.4: Using Scanner Class
Demo : How to use Scanner class?

Execute

- ScannerDemo.java program
- ParseString.java program

5.5: The System Class
The System Class

Used to interact with any of the system resources

Cannot be instantiated

Contains a methods and variables to handle system I/O

System class provides facilities like Standard input, Standard output and Error output streams

Method	Description
`void currentTimeMillis()`	Returns the current time in terms of milliseconds since midnight, January 1, 1970
`void gc()`	Initiates the garbage collector.
`void exit(int code)`	Halts the execution and returns the value of integer to parent process usually to an operating system.

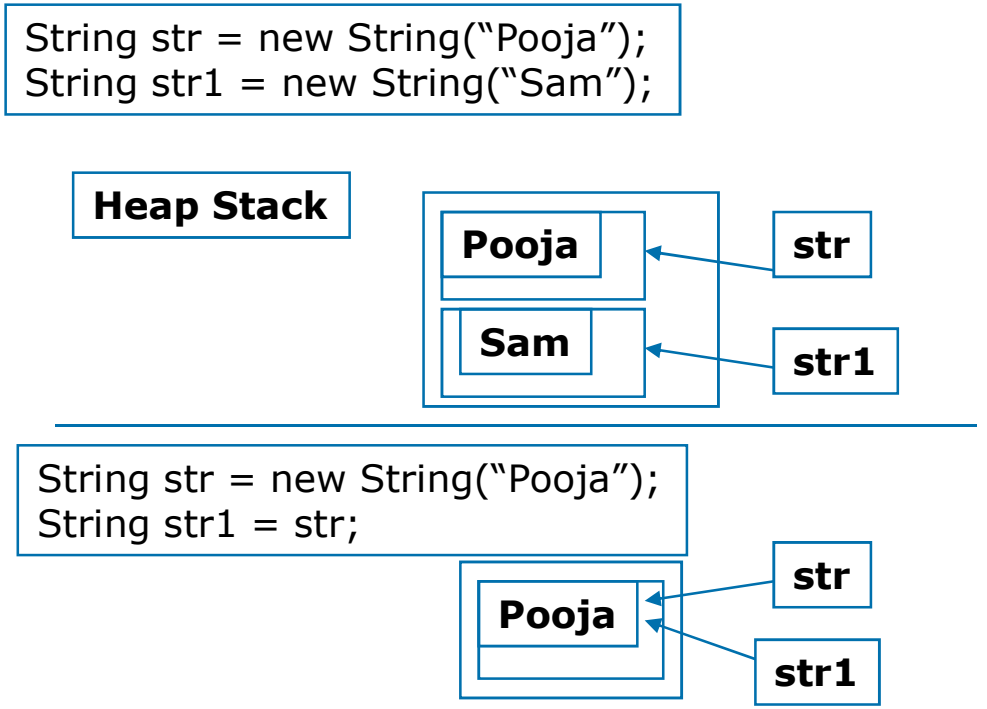
5.5: The System Class Demo

Execute the Elapsed.java program

5.6: String Handling
String Handling

String is handled as an object of class String and not as an array of characters

- String class is a better & convenient way to handle any operation
- String objects are immutable



5.6: String Handling
Important Methods

length(): length of string

indexOf(): searches an occurrence of a char, or string within other string

substring(): Retrieves substring from the object

trim(): Removes spaces

valueOf(): Converts data to string

isEmpty(): Added in Java 6 to check whether string is empty or not

concat(String s) : Used to concatenate a string to an existing string. Eg

```
String string = "Core ";
System.out.println( string=string.concat(" Java") );
Output -> "Core Java”
```

5.6: String Handling
String Concatenation

Use a “+” sign to concatenate two strings Examples:

```
Example: String string = "Core " + "Java";    -> Core Java
```

- String concatenation operator if one operand is a string:

```
String a = "String"; int b = 3; int c=7
System.out.println(a + b + c);  -> String37
```

- Addition operator if both operands are numbers:

```
System.out.println(a + (b + c)); -> String10
```

5.6: String Handling
String Comparison

Output : Hello equals Hello -> true
Hello == Hello -> false

```
class EqualsNotEqualTo {
   public static void main(String args[]) {
        String str1 = "Hello";
        String str2 = new String(str1);
        System.out.println(str1 + " equals " + str2 + " -> " +
                    str1.equals(str2));
        System.out.println(str1 + " == " + str2 + " -> " + (str1 ==str2));
   }
}
```

5.6: String Handling
StringBuffer Class

Following classes allow modifications to strings:

- java.lang.StringBuffer
- java.lang.StringBuilder

Many string object manipulations end up with a many abandoned string objects in the String pool, since String objects are immutable

```
StringBuffer sb = new StringBuffer("abc");
sb.append("def");
System.out.println("sb = " + sb); // output is "sb = abcdef"
```

5.6: String Handling
StringBuilder Class

Added in Java 5

Exactly the same API as the *StringBuffer* class, except:

- It is not thread safe
- It runs faster than StringBuffer

```
StringBuilder sb = new StringBuilder("abc");
sb.append("def").reverse().insert(3, "---");
System.out.println( sb ); // output is "fed---cba"
```

5.6: String Handling
Demo

Execute the following programs:

- SimpleString.java
- ToStringDemo.java
- StringBufferDemo.java
- CharDemo.java

5.7: Date and Time API
Date and Time API

Added in Java SE 8 under java.time package.

Enhanced API to make extremely easy to work with Date and Time.

Immutable API to store date and time separately.

- Instant
- LocalDate
- LocalTime
- LocalDateTime
- ZonedDateTime

It has also added classes to measure date and time amount.

- Duration
- Period

Improved way to represent units like day and months.

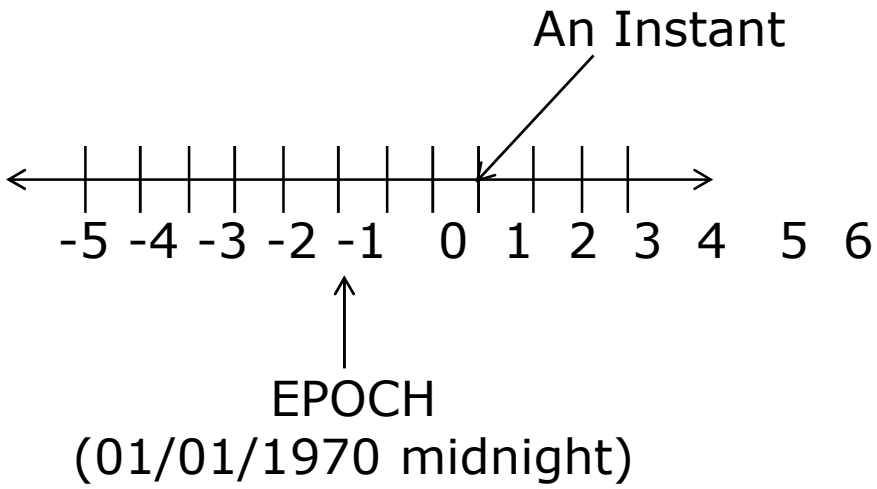
Generalised parsing and formatting across all classes.

5.7: Date and Time API
The Instant Class

An object of instant represent point on the time line.

The reference point is the standard java epoch.

This class is useful to represent machine timestamp.




```
Instant currentTime = Instant.now();
```

The static method "now" of Instant class is used to represent current time.

5.7: Date and Time API
The LocalDate Class

It represent date without time and zone.

Useful to represent date events like birthdate .

Following table shows important methods of LocalDate:

Method	Uses
now	A static method to return today's date.
of	Creates local date from year, month and date.
getXXX ()	Used to return various part of date.
plusXXX()	Add the specified factor and return a LocalDate.
minusXXX()	Subtracts the specified factor and return a LocalDate.
isXXX()	Performs checks on LocalDate and returns Boolean value.
withXXX()	Returns a copy of LocalDate with the factor set to the given value.

5.7: Date and Time API
The ZonedDateTime Class

It stores all date and time fields, to a precision of nanoseconds, as well as a time-zone and zone offset.

Useful to represent arrival and departure time in airline applications.

Following table shows important methods of ZonedDateTime:

Method	Uses
now	A static method to return today's date.
of	Overloaded static method to create zoned date time object.
getXXX ()	Used to return various part of ZonedDateTime.
plusXXX()	Add the specified factor and return a ZonedDateTime.
minusXXX()	Subtracts the specified factor and return a ZonedDateTime.
isXXX()	Performs checks on ZonedDateTime and returns Boolean value.
withXXX()	Returns ZonedDateTime with the factor set to the given value.

5.7: Date and Time API
Period and Duration

The Period class models a date-based amount of time, such as five days, a week or three years.

Duration class models a quantity or amount of time in terms of seconds and nanoseconds. It is used represent amount of time between two instants.

Following table shows important and common methods of both:

Method	Uses
between	Use to create either Period or Duration between LocalDates.
of	Creates Period/Duration based on given year, months & days.
ofXXX()	Creates Period/Duration based on specified factors.
getXXX ()	Used to return various part of Period/Duration.
plusXXX()	Add the specified factor and return a LocalDate.
minusXXX(	Subtracts the specified factor and return a LocalDate.
isXXX()	Performs checks on LocalDate and returns Boolean value.
withXXX()	Returns a copy of LocalDate with the factor set to the given value.

5.7: Date and Time API
Formatting and Parsing Date and Time

Java SE 8 adds DateTimeFormatter class which can be used to format and parse the date and time.

To either format or parse, the first step is to create instance of DateTimeFormatter.

Following are few important methods available on this to create DateTimeFormatter.

Method	Uses
ofLocalizedDate(dateStyle)	Date style formatter from locale
ofLocalizedTime(timeStyle)	Time style formatter from locale
ofLocalizedDateTime(dateTimeStyle)	Date and time style formatter from locale
ofPattern(StringPattern)	Custom style formatter from string

Once formatter object created, parsing/formatting is done by using parse() and format() methods respectively. These methods are available on all major date and time classes .

5.7: Date and Time API
Demo

Execute the following programs:

- LocalDateDemo.java
- ZonedDateTimeDemo.java
- CalculatingPeriod.java
- FormattingDate.java
- ParsingDate.java

5.8: Best Practices
Best Practices - String Handling

Use StringBuffer for appending

String.charAt() is slow

Use String.intern method to improve performance

Use isEmpty() method to check empty string in faster way

Notes Page

5.8: Best Practices
Common Best Practices (contd..)

Assert is for private arguments only

Validate method arguments

Fields should usually be private

Instance variable should not be used directly in a method

Do not use *valueOf* to convert to primitive type

Downward cast is costly

Notes Page

5.9: Exploring Java Basics Lab

Lab 3: Exploring Basic Java Class Libraries

Summary

In this lesson you have learnt:

- The Object Class
- Wrapper Classes
- Type casting
- Using Scanner Class
- The System Class
- String Handling
- Date and Time API
- Best Practices

Review Questions

Question 1: String objects are mutable and thus suitable to use if you need to append or insert characters into them.

- True/False

Question 2: Which of the following static fields on wrapper class indicates range of values for its class:

- Option 1:MIN_VALUE
- Option 2: MAX_VALUE
- Option 3: SMALL_VALUE
- Option 4: LARGE_VALUE