Core Java

Day 09 Agenda

- Deep Copy vs Shallow Copy
- String class
- StringBuffer vs StringBuilder class
- AutoCloseable interface
- ~~Exception Handling~~

Java Strings

- java.lang.Character is wrapper class that represents char. In Java, each char is 2 bytes because it follows unicode encoding.
- String is sequence of characters.
  1. java.lang.String: "Immutable" character sequence
  2. java.lang.StringBuffer: Mutable character sequence (Thread-safe)
  3. java.lang.StringBuilder: Mutable character sequence (Not Thread-safe)
- String helpers
  1. java.util.StringTokenizer: Helper class to split strings

String objects

- java.lang.String is class and strings in java are objects.
- String constants/literals are stored in string pool.

```
String str1 = "Sunbeam";
```

- String objects created using "new" operator are allocated on heap.

```
String str2 = new String("Nilesh");
```

- In java, String is immutable. If try to modify, it creates a new String object on heap.

String literals

- Since strings are immutable, string constants are not allocated multiple times.
- String constants/literals are stored in string pool. Multiple references may refer the same object in the pool.
- String pool is also called as String literal pool or String constant pool.
- From Java 7, String pool is in the heap space (of JVM).
- The string literal objects are created during class loading.

String objects vs String literals

- Example 01:

```
String s1 = "Sunbeam";
String s2 = "Sunbeam";
System.out.println(s1 == s2);       // ???
System.out.println(s1.equals(s2));  // ???
```

- Example 02:

```
String s1 = new String("Sunbeam");
String s2 = new String("Sunbeam");
System.out.println(s1 == s2);       // ???
System.out.println(s1.equals(s2));  // ???
```

- Example 03:

```
String s1 = "Sunbeam";
String s2 = new String("Sunbeam");
System.out.println(s1 == s2);       // ???
System.out.println(s1.equals(s2));  // ???
```

- Example 04:

```
String s1 = "Sunbeam";
String s2 = "Sun" + "beam";
System.out.println(s1 == s2);       // ???
System.out.println(s1.equals(s2));  // ???
```

- Example 05:

```
String s1 = "Sunbeam";
String s2 = "Sun";
String s3 = s2 + "beam";
System.out.println(s1 == s3);       // ???
System.out.println(s1.equals(s3));  // ???
```

- Example 06:

```
String s1 = "Sunbeam";
String s2 = new String("Sunbeam").intern();
System.out.println(s1 == s2);       // ???
System.out.println(s1.equals(s2));  // ???
```

- Example 07:

```
String s1 = "Sunbeam";
String s2 = "SunBeam";
System.out.println(s1 == s2);       // ???
System.out.println(s1.equals(s2));  // ???
System.out.println(s1.equalsIgnoreCase(s2));  // ???
System.out.println(s1.compareTo(s2));  // ???
System.out.println(s1.compareToIgnoreCase(s2));  // ???
```

String operations

- int length()
- char charAt(int index)
- int compareTo(String anotherString)
- boolean equals(String anotherString)
- boolean equalsIgnoreCase(String anotherString)
- boolean matches(String regex)
- boolean isEmpty()
- boolean startsWith(String prefix)
- boolean endsWith(String suffix)
- int indexOf(int ch)
- int indexOf(String str)
- String concat(String str)
- String substring(int beginIndex)
- String substring(int beginIndex, int endIndex)
- String[] split(String regex)
- String toLowerCase()
- String toUpperCase()
- String trim()
- byte[] getBytes()
- char[] toCharArray()
- String intern()
- static String valueOf(Object obj)
- static String format(String format, Object... args)

StringBuffer vs StringBuilder

- StringBuffer and StringBuilder are final classes declared in java.lang package.
- It is used create to mutable string instance.
- equals() and hashCode() method is not overridden inside it.
- Can create instances of these classes using new operator only. Objects are created on heap.
- StringBuffer capacity grows if size of internal char array is less (than string to be stored).
  - The default capactiy is 16.

```
int max = (minimumCapacity > value.length? value.length * 2 + 2 :
value.length);
minimumCapacity = (minimumCapacity < max? max : minimumCapacity);
char[] nb = new char[minimumCapacity];
```

- StringBuffer implementation is thread safe while StringBuilder is not thread-safe.
- StringBuilder is introduced in Java 5.0 for better performance in single threaded applications.

Examples

- Example 01:

```
StringBuffer s1 = new StringBuffer("Sunbeam");
StringBuffer s2 = new StringBuffer("Sunbeam");
System.out.println(s1 == s2);        // false
System.out.println(s1.equals(s2));   // false
```

- Example 02:

```
StringBuffer s1 = new StringBuffer("Sunbeam");
String s2 = new String("Sunbeam");
System.out.println(s1 == s2);        // false
System.out.println(s1.equals(s2));   // false
```

- Example 03:

```
String s1 = new String("Sunbeam");
StringBuffer s2 = new StringBuffer("Sunbeam");
System.out.println(s1.equals(s2));   // false -- String compared with
StringBuffer
System.out.println(s1.equals(s2.toString()));   // true -- String compared
with String
```

- Example 04:

```
StringBuffer s1 = new StringBuffer("Sunbeam");
StringBuffer s2 = s1.reverse();
System.out.println(s1 == s2);        // true
System.out.println(s1.equals(s2));   // true
```

- Example 05:

```
StringBuilder s1 = new StringBuilder("Sunbeam");
StringBuilder s2 = new StringBuilder("Sunbeam");
System.out.println(s1 == s2);       // false
System.out.println(s1.equals(s2));  // false -- calls Object.equals()
```

- Example 06:

```
StringBuffer s = new StringBuffer();
System.out.println("Capacity: " + s.capacity() + ", Length: " + s.length());
// 16, 0
s.append("1234567890");
System.out.println("Capacity: " + s.capacity() + ", Length: " + s.length());
// 16, 10
s.append("ABCDEFGHIJKLMNOPQRSTUV");
System.out.println("Capacity: " + s.capacity() + ", Length: " + s.length());
// 34, 32
```

StringTokenizer

- Used to break a string into multiple tokens - like split() method.
- Methods of java.util.StringTokenizer
  - int countTokens()
  - boolean hasMoreTokens()
  - String nextToken()
  - String nextToken(String delim)
- Example:

```
String str = "My name is Bond, James Bond.";
String delim = " ,.";
StringTokenizer tokenizer = new StringTokenizer(str, delim);
while(tokenizer.hasMoreTokens()) {
    String token = tokenizer.nextToken();
    System.out.println(token);
}
```

Resource Management

- System resources should be released immediately after the use.
- Few system resources are Memory, File, IO Devices, Socket/Connection, etc.
- The Garbage collector automatically releases memory if objects are no more used (unreferenced).
- The GC collector doesn't release memory/resources immediately; rather it is executed only memory is full upto a threshold.
- The standard way to release the resources immediately after their use is java.io.Closeable interface. It has only one method.
  - void close() throws IOException;

- Programmer should call close() explicitly on resource object after its use.
    - e.g. FileInputStream, FileOutputStream, etc.
- Java 7 introduced an interface java.lang.AutoCloseable as super interface of Closeable. It has only one method.
    - void close() throws Exception;
- Since it is super-interface of Closeable, all classes implementing Closeable now also inherit from AutoCloseable.
- If a class is inherited from AutoCloseable, then it can be closed using try-with-resource syntax.

```
class MyResource implements AutoCloseable {
    // ...
    public void close() {
        // cleanup code
    }
}

class Program {
    public static void main(String[] args) {
        try(MyResource res = new MyResource()) {
            // ...
        } // res.close() called automatically
    }
}
```

- The Scanner class is also AutoCloseable.

```
class Program {
    public static void main(String[] args) {
        try(Scanner sc = new Scanner(System.in)) {
            // ...
        } // sc.close() is auto-closed
    }
}
```