Overview from Java 9

Content

- 1) Compact Strings
- 2) Logging and Service
- 3) Collection Static Factory Methods

COMPACT STRINGS

```java
public class HelloWorldStringConcatComplex
{
   public static void main(final String[] arguments)
   {
      String message = "Hello";
      for (int i=0; i<25; i++)
      {
         message += i;
      }
      out.println(message);
   }
}
```

```
Classfile /C:/java/examples/helloWorld/classes/dustin/examples/HelloWorldStringConcatComplex.class
  Last modified Jan 30, 2019; size 766 bytes
  MD5 checksum 772c4a283c812d49451b5b756aef55f1
  Compiled from "HelloWorldStringConcatComplex.java"
public class dustin.examples.HelloWorldStringConcatComplex
  minor version: 0
  major version: 52
     . . .
  public static void main(java.lang.String[]);
    descriptor: ([Ljava/lang/String;)V
    flags: ACC_PUBLIC, ACC_STATIC
    Code:
      stack=2, locals=3, args_size=1
         0: ldc           #2                  // String Hello
         2: astore_1
         3: iconst_0
         4: istore_2
         5: iload_2
         6: bipush        25
         8: if_icmpge     36
        11: new           #3                  // class java/lang/StringBuilder
        14: dup
        15: invokespecial #4                  // Method java/lang/StringBuilder."19: invokevirtual #5
        22: iload_2
        23: invokevirtual #6                  // Method java/lang/StringBuilder.append:(I)Ljava/lang/Stri
        26: invokevirtual #7                  // Method java/lang/StringBuilder.toString:()Ljava/lang/Str
        29: astore_1
        30: iinc          2, 1
":()V 18: aload_1 33: goto 5 36: getstatic #8 // Field java/lang/System.out:Ljava/io/PrintStream; 39: alo
```

```
Classfile /C:/java/examples/helloWorld/classes/dustin/examples/HelloWorldStringConcatComplex.class
  Last modified Jan 30, 2019; size 1018 bytes
  MD5 checksum 967fef3e7625965ef060a831edb2a874
  Compiled from "HelloWorldStringConcatComplex.java"
public class dustin.examples.HelloWorldStringConcatComplex
  minor version: 0
  major version: 55
     . . .
  public static void main(java.lang.String[]);
    descriptor: ([Ljava/lang/String;)V
    flags: (0x0009) ACC_PUBLIC, ACC_STATIC
    Code:
      stack=2, locals=3, args_size=1
         0: ldc           #2                  // String Hello
         2: astore_1
         3: iconst_0
         4: istore_2
         5: iload_2
         6: bipush        25
         8: if_icmpge     25
        11: aload_1
        12: iload_2
        13: invokedynamic #3,  0              // InvokeDynamic #0:makeConcatWithConstants:(Ljava/lang/Str
        18: astore_1
        19: iinc          2, 1
        22: goto          5
        25: getstatic     #4                  // Field java/lang/System.out:Ljava/io/PrintStream;
        28: aload_1
        29: invokevirtual #5                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        32: return
```

LOGGING AND SERVICE

- JEP 264 Platform Logging and Service
- 1.java.util.ServiceLoader API, LoggerFinder implementation is located and loaded using the system class loader.
- 2.If no concrete implementation is found, the JDK internal default implementation of the LoggerFinder service is used.
- 3.The default implementation of the service uses java.util.logging as a backend when the java.logging module, by default, log messages are routed to java.util.logging.Logger as before.
- 4.LoggerFinder service makes it possible for an application/framework to plug in its own external logging backend, without needing to configure both java.util.logging and that backend.

```
package java.lang;

...

public class System {

    System.Logger getLogger(String name) { ... }

    System.Logger getLogger(String name, ResourceBundle bundle) { ... }

}
```

COLLECTION STATIC FACTORY METHODS

- JEP 269: Convenience Factory Methods for Collections
- **Motivation**
- Makes it easier to create instances of collections and maps with small numbers of elements. New static factory methods on the List, Set, and Map interfaces make it simpler to create immutable instances of those collections.
- Set<String> alphabet = Set.of("a", "b", "c");
- 1.They are structurally immutable. Elements cannot be added or removed. Calling any mutator method will always cause UnsupportedOperationException to be thrown. However, if the contained elements are themselves mutable, this may cause the Set to behave inconsistently or its contents to appear to change.
- 2.They disallow null elements. Attempts to create them with null elements result in NullPointerException.
- 3.They are serializable if all elements are serializable.
- 4.They reject duplicate elements at creation time. Duplicate elements passed to a static factory method result in IllegalArgumentException.
- 5.The iteration order of set elements is unspecified and is subject to change.
- 6.They are value-based. Callers should make no assumptions about the identity of the returned instances. Factories are free to create new instances or reuse existing ones. Therefore, identity-sensitive operations on these instances (reference equality (==), identity hash code, and synchronization) are unreliable and should be avoided.

```java
public class ListTest {
    public static void main(String[] args) {
        // Create few unmodifiable lists
        List<Integer> emptyList = List.of();
        List<Integer> luckyNumber = List.of(19);
        List<String> vowels = List.of("A", "E", "I", "O", "U");

        System.out.println("emptyList = " + emptyList);
        System.out.println("singletonList = " + luckyNumber);
        System.out.println("vowels = " + vowels);

        try {
            // Try using a null element
            List<Integer> list = List.of(1, 2, null, 3);
        } catch(NullPointerException e) {
            System.out.println("Nulls not allowed in List.of().");
        }

        try {
            // Try adding an element
            luckyNumber.add(8);
        } catch(UnsupportedOperationException e) {
            System.out.println("Cannot add an element.");
        }

         try {
            // Try removing an element
            luckyNumber.remove(0);
        } catch(UnsupportedOperationException e) {
            System.out.println("Cannot remove an element.");
        }
    }
}
```