Evolution of Java Language

Java 1.1	❑ Nested classes
	❑ JDBC
	❑ Java Beans
	❑ Char Streams
Java 1.2	❑ Collections Framework, Swing API
Java 1.3	
Java 1.4	❑ Assertions
Java 5	❑ Auto-boxing and Auto-unboxing
	❑ Annotations
	❑ Generics
	❑ Enhanced for loop
	❑ Enumeration
	❑ Static import
	❑ Variable arguments

Evolution of Java Language

Java 6

- ❑ Scripting API

Java 7

- ❑ Try with resources
- ❑ Underscore in numeric literals and binary literals
- ❑ Diamond operator (<>)
- ❑ Support for strings in switch
- ❑ Multiple exceptions in catch block
- ❑ More precise rethrow
- ❑ New IO

Java 8

- ❑ Default and static methods in Interface
- ❑ Lambda Expressions
- ❑ Streams
- ❑ New Date API

Evolution of Java Language

Java 9

- ❑ Private methods in interface
- ❑ Enhancements to try with resources
- ❑ Compact Strings
- ❑ JShell
- ❑ New factory methods in Collections
- ❑ New methods in Streams
- ❑ New methods in LocalDate
- ❑ JPMS

Java 10

- ❑ Local variables Type Inference

Evolution of Java Language

Java 11

- ❑ Added new methods to String class
- ❑ Added new methods to Files class and Path interface
- ❑ Lambdas can use var for parameters
- ❑ Java can run .java file directly without using Javac

Java 12

- ❑ New methods to String class
- ❑ Compact Number Format
- ❑ Enhancements to switch

Java 13

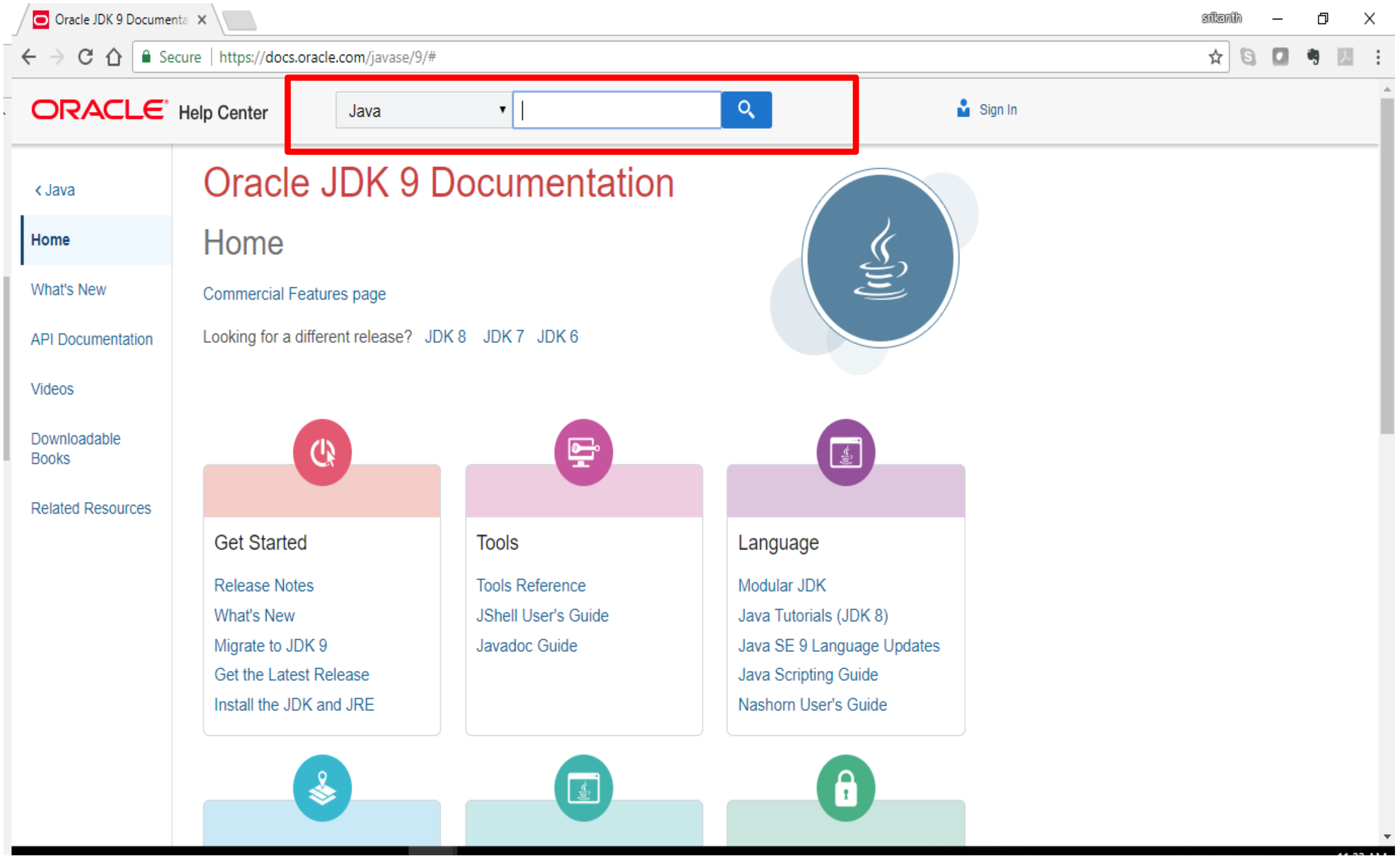
- ❑ Text Blocks and related String methods
- ❑ Switch Expressions changes from 12

What’s New In Java 9

- ❑ Search facility in documentation
- ❑ Jshell (REPL)
- ❑ Language Enhancements
- ❑ New factory methods in Collections
- ❑ Stream API Improvements
- ❑ Compact Strings
- ❑ New methods in Date API
- ❑ Java Platform Module System (JPMS)

Released in Sep, 2017

Search facility in Java 9 Documentation



Oracle JDK 9 Documenta

https://docs.oracle.com/javase/9/#

ORACLE Help Center

Java

Sign In

< Java

Home

What's New

API Documentation

Videos

Downloadable Books

Related Resources

Oracle JDK 9 Documentation

Home

Commercial Features page

Looking for a different release? JDK 8 JDK 7 JDK 6

Get Started

Release Notes

What's New

Migrate to JDK 9

Get the Latest Release

Install the JDK and JRE

Tools

Tools Reference

JShell User's Guide

Javadoc Guide

Language

Modular JDK

Java Tutorials (JDK 8)

Java SE 9 Language Updates

Java Scripting Guide

Nashorn User's Guide

jShell – Java Shell

- Also known as REPL (Read Evaluate Print Loop)
- Used to execute any java code from command prompt
- Allows small code to be executed without having to create a class and main method
- Start by using jshell at command prompt
- We can give any Java statement or expression at jshell prompt (jshell>)
- Results of expressions are stored in variable named as $n where n is a number. These variables are called as internal variables
- Code can be typed in multiple lines
- Commands of jshell start with /. For example, /help and /exit
- It supports TAB for completion of expression
- It automatically imports some packages. We can get the list of packages imported using /import command

jshell – commands

/drop	Drops a snippet
/edit	Opens an editor
/exit	Exits jshell
/history	Displays what was entered in this session
/help	Displays information about commands and subjects
/imports	Displays active imports
/list	Displays list of snippets and their ID
/methods	Lists methods that were created
/open file	Opens the file and reads snippet into jshell. In place of file you can give DEFAULT, JAVASE or PRINTING.
/reset	Resets the session
/save file	Saves snippets and commands to the file specified
/set	Sets configuration information
/types	Displays types created in session
/vars	Displays variables created in session

jshell – shortcuts

Tab Completion

When entering snippets, commands, subcommands, command arguments, or command options, use the Tab key to automatically complete the item. If the item can’t be determined from what was entered, then possible options are provided.

Command abbreviations

An abbreviation of a command is accepted if the abbreviation uniquely identifies a command.

History navigation

A history of what was entered is maintained across sessions. Use the up and down arrows to scroll through commands and snippets from the current and past sessions.

History search

Use the Ctrl+R key combination to search the history for the string entered. The prompt changes to show the string and the match. Ctrl+R searches backwards from the current location in the history through earlier entries. Ctrl+S searches forward from the current location in the history through later entries.

Defining Method

```
jshell> int add(int n1, int n2)
   ...> {
   ...>     return n1 + n2;
   ...> }
|   created method add(int,int)

Jshell>
```

```
jshell> /methods
|     int add(int,int)
```

Defining Class

```
jshell> class Person{
   ...>  private String name;
   ...>  private int age;
   ...>  public Person(String name, int age) {
   ...>   this.name=name;
   ...>   this.age=age;
   ...>  }
   ...>  public String toString() {
   ...>   return  name + " : "  + age;
   ...>  }
   ...> }
|  created class Person
jshell> /types
|    class Person

Jshell>
```

Saving and Loading

```
>jshell
|  Welcome to JShell -- Version 12.0.1
|  For an introduction type: /help intro

jshell> /open demo.jsh

jshell> /methods
|    double getNetPrice(double)

jshell> /var
|    double TAX = 12.5
|    double $3 = 13500.0

jshell>
```

Language Enhancements – Milling Project Coin

- ❑ **Private methods in interface**
- ❑ **Enhancements to try with resources**

Interface Extensions – Private Methods

- ❑ Interface can now have private methods that can be called from other methods of the interface
- ❑ Private methods in interface cannot be called from implementing class
- ❑ Provides more reusability in interfaces as these private methods can be called from other default methods
- ❑ Must be declared with private keyword and must contain body

Private Methods in Interface

```
public interface Logger {
 private void log(String message, String target) {
     // log message to target
 }

 default void logToFile(String message) {
     log(message, "file");
 }

 default void logToDatabase(String message) {
     log(message, "database");
 }
}
```

Try with Resources Enhancements

- Try with resource can now use a resource that is already declared outside the Try-With-Resource Statement as final or effectively final

```
// Before Java 9
Scanner s1 = new Scanner(System.in);
// some other code
try (Scanner s2 = s1) {
      String line = s2.nextLine();
}
```

```
// From Java 9
Scanner s = new Scanner(System.in);

// Variable s can be used with try with resource
// even through it was created earlier.

try (s) {
      String line = s.nextLine();
}
```

New Methods in List, Set, Map and Map.Entry

- ❑ List.of()
- ❑ List.of(values)
- ❑ Set.of()
- ❑ Set.of(values)
- ❑ Map.of()
- ❑ Map.of(key, value, key, value,…)

List.of()

```
// Immutable Empty List
List<String>  emptyList = List.of();

// Java 8
List<String> list = new ArrayList<>();
list.add("one");
list.add("two");
list.add("three");
List<String>
immutableList=Collections.unmodifiableList(list);

// Java 9
List<String> immutableList=List.of("one","two","three");
```

Immutable List Features

- List is immutable i.e. we can’t add, delete or modify elements
- Nulls are not allowed
- Serializable if all elements are serializable
- We can create a list of up to 10 elements as there are 10 parameters for of() method
- List.<E>of(E…elements) can take any number of parameters including an array

Map.of() and Map.entry()

```
// Immutable Empty Map
Map<String, String> emptyMap = Map.of();

// Immutable Map
Map<String,String> phones=
     Map.of("Bill","39393393988","Larry","9988776655");

// Immutable Map
Map<Integer,String> map =
 Map.ofEntries( Map.entry(1,"One"),Map.entry(2,"Two"));
```

Stream API Improvements

Stream API added new methods to Stream Interface

- ❑ **takeWhile()**
- ❑ **dropWhile()**
- ❑ **iterate()**

takeWhile()

default Stream<T> takeWhile(Predicate<? super T> predicate)
Returns elements of stream that match predicate. It stops taking elements when predicate is false.

```
Stream<Integer> nums = Stream.of(10,22,31,34,87,12);
Stream<Integer> evenNums =
         nums.takeWhile( n -> n % 2 == 0 );
evenNums.forEach(System.out::println);
```

```
Output
======
10
22
```

dropWhile()

default Stream<T> dropWhile(Predicate<? super T> predicate)

Returns elements of stream after dropping elements that match predicate. It stops dropping elements when predicate is false.

```
Stream<Integer> nums = Stream.of(10,22,31,34,87,12);
Stream<Integer> nums2 = nums.dropWhile(n -> n % 2 == 0);
nums2.forEach(System.out::println);
```

```
Output
======
31
34
87
12
```

iterate()

static <T> Stream<T> iterate(T seed, Predicate<? super T> hasNext, UnaryOperator<T> next)

Returns a stream of elements that are produced by **next** until **hasNext** is false.

```
Stream<Integer> oddNums =
      Stream.iterate(1,v->v < 10,v->v + 2);
oddNums.forEach(System.out::println);
```

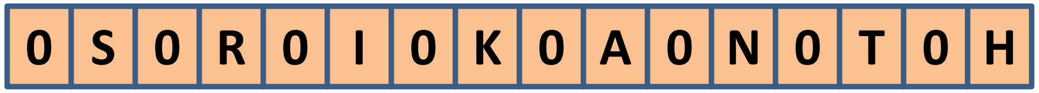
```
Output
======
1
3
5
7
9
```

Compact Strings

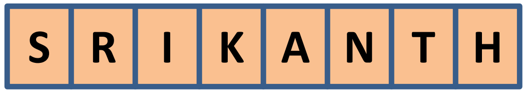
- Compact string is a string that uses only one byte for character instead of 2 bytes as it is the case in Java 8
- Java decides whether to use byte[] or char[] to store strings based on character set used
- For LATIN-1 representation, a byte array is used and in other cases char array is used (UTF-16 representation)
- To know whether a string is using char[] or byte[] , a field called coder is used internally
- The whole implementation is automatic and transparent to developer
- Compact strings reduce the amount of memory occupied by strings by half

Compact Strings

String in Java 8



String in Java 9



New Methods in LocalDate class

Stream<LocalDate> datesUntil (LocalDate endExclusive)
Stream<LocalDate> datesUntil (LocalDate endExclusive, Period step)

```
// Print all dates from 1-10-2017 to 31-12-2017

LocalDate start = LocalDate.of(2017, 10, 1);
start.datesUntil(LocalDate.of(2018,1,1))
     .forEach(System.out::println);
```

New Features of Java 10

- ❑ Released in March, 2018
- ❑ Local-variable type inference
- ❑ Garbage collector (GC) enhancements

Local-variable Type Inference (JEP 286)

- Java infers type of a local variable based on right hand side (RHS) expression in initialization.
- Used with new keyword var.

```
// Java 9 and before
ArrayList<String> names = new ArrayList<String>();

// Java 10
var names = new ArrayList<String>();
var filename = “tempfile.txt”;
var size = 10;
```

Java 11

- Released on 25th Sep 2018
- Added methods to String class
- Added methods to Files class and Path interface
- Lambdas can use var for parameters
- Java can run .java file directly without using Javac
- In Windows and macOS, installing the JDK in previous releases optionally installed a JRE. In JDK 11, this is no longer an option.
- Auto-update, which was available for JRE installations on Windows and macOS, is no longer available.
- JavaFX is no longer included in the JDK

New String Methods

- ❑ boolean isBlank()
- ❑ Stream lines()
- ❑ String repeat(int count)
- ❑ String strip()
- ❑ String stripLeading()
- ❑ String stripTrailing()

New String methods in Java 11

```java
String st = "Hello!";
System.out.println( st.repeat(5));
System.out.println("  ".isBlank());
System.out.println(" Java 11  ".strip());

String wish = "Hello!\nHow are you?";
wish.lines().forEach(System.out::println);
```

```
Hello!Hello!Hello!Hello!Hello!
true
Java 11
Hello!
How are you?
```

New Methods in Files class

- static String readString(Path)
- static Path writeString(Path path, CharSequence csq)

```
import java.nio.file.Files;
import java.nio.file.Path;

public class NewFilesMethods {
  public static void main(String[] args) throws Exception {
    Path path = Path.of("c:\\dev\\java\\names.txt");
    Files.writeString(path, "First\nSecond\nThird");
    System.out.println(Files.readString(path));
  }
}
```

New Methods in Path interface

- static Path of(String first, String... more)
- static Path of(URI uri)

Run .java directly

Hello.java

```java
public class Hello {
  public static void main(String args[]) {
     System.out.println("Hello!");
  }
}
```

```
C:\dev\java>Java Hello.java
```

What’s new in Java 12

- ❑ New methods in String class
- ❑ Compact Number formats
- ❑ Files.mismatch() method
- ❑ Enhancements to switch statement

The indent() in String class

indent(size)

Adjusts the indentation of each line of this string based on the value of n, and normalizes line termination characters.

```
jshell> "Hello".indent(10)
$1 ==> "          Hello\n"
```

```
jshell> "First\nSecond\nThird".indent(5).lines()
                              .forEach(System.out::println)
     First
     Second
     Third
```

The transform() in String class

```
Public <R> R transform(Function<? super String,
                                ? extends R> f)
```

- ❑ This method allows the application of a function to this string.
- ❑ The function should expect a single String argument and produce an R result.
- ❑ Any exception thrown by f() will be propagated to the caller.

```
jshell> var name = "Srikanth"
jshell> name.transform(String::toUpperCase)
              .transform(s -> s.substring(0,5))
$5 ==> "SRIKA"
```

Compact Number Format

- Method getCompactNumberInstance() of NumberFormat class returns an object that can be used to format numbers to get compact value of the given number.
- It returns 1K for 1000, and 1M for 1000000 etc.

```
jshell> import java.text.NumberFormat
jshell> NumberFormat nf =
NumberFormat.getCompactNumberInstance()
jshell> nf.format(1000)
$9 ==> "1K"
jshell> nf.format(1000000)
$10 ==> "1M"
```

Files.mismatch()

Finds and returns the position of the first mismatched byte in the content of two files, or -1L if there is no mismatch.

```
public static long mismatch(Path path, Path path2)
                    throws IOException
```

```
jshell> Files.mismatch(Paths.get("names.txt"),
                       Paths.get("names2.txt"))
$10 ==> -1
```

The switch expression

- ❑ Switch now supports multiple labels
- ❑ Switch can be used either as statement or as an expression
- ❑ No need to use break statement as there is no fall-through

```
switch(code)
{
         case 1     -> System.out.println(10);
         case 2, 3  -> System.out.println(20);
         default    -> System.out.println(5);
};
```

```
// Switch expression
disrate = switch(code) {
            case 1     -> 10;
            case 2, 3  -> 20;
            default    -> 5;
};
```

```
java --enable-preview  --source 12 Test.java
```

What’s new in Java 13

- ❑ Text blocks
- ❑ Changes to switch

Text Blocks

- A text block is a multi-line string literal that avoids the need for most escape sequences, automatically formats the string in predictable ways, and gives the developer control over format when desired.
- It may be used to denote a string anywhere that a string literal may be used but offers greater expressiveness.

```
String st = """
       First line
       Second Line
       Third line
       """;

System.out.println(st);
```

Changes to switch

- New statement yield is used to return value from a switch expression.

```
int disrate =
   switch(code) {
     case 1:
          yield 10;
     case 2, 3:
          yield 20;
     default:
          yield 5;
   };

System.out.println(disrate);
```