Java - 1991 - James Gosling & Team
- Green Team
- Sun Micro System

Java - Embedded systems
- "Compile Once Run Everywhere"
- First product : *7
Universal Remote Control.

Internet - Late 80's - web pages - sharing info.
- 1990 - www
- 1995 - Java applets - interactive

Java — Compilation and Execution flow

JDK vs JRE vs JVM

SDK = Software Development Kit
= compiler/linker/debugger + libraries + documentation + IDE (optional)
(dev tools)

JDK = Java's SDK = dev tools + libraries + java docs
(javac, java) (rt.jar)

Core Java

Course Introduction

Course contents -- Java 8

- Language features (Installation, Buzzwords, History, JRE/JDK/JVM, Compilation, Operators, Data types, Narrowing/Widening)
- Control Structures (if-else, loops, switch, recursion, wrapper classes, boxing, value/reference type)
- Input/Output: Commandline args, Scanner, Console
- OOP basics (OOP pillars, class/object, class methods, ctor chaining, overloading, final field, toString()), enum
- Static (static fields/method/block), Singleton, Package (classpath), static import
- Arrays (1-D & 2-D array, primitive and object array, java.util.Arrays)
- OOP advanced (aggregation/association/composition, Inheritance, super, Method overriding, instanceof, final method/class, Abstract class, Java 7 Interfaces, Marker interfaces)
- Java classes (java.lang.Object, Date/LocalDate/Calendar, String/StringBuilder/StringBuffer, etc)
- Exception Handling (try-catch-throw, throws, finally, try-with-resource/Closeable, errors, custom exception, checked/unchecked, overriding rules)
- Java Generics (methods, classes, interfaces), Comparable/Comparator interface, Java 8 Interfaces
- Nested classes, Functional Interfaces, Lambda expressions, Method references
- Java Collections (Hierarchy, Lists, Iterator, Sets, Maps, Queue/Stack, Fail-safe/Fail-fast Iterators, Legacy collections, Collections)
- Functional programming (Concepts, Java 8 interfaces, Stream characteristics, Stream operations, Collectors)
- Java IO (Binary vs Text streams, File IO, Stream chaining, Data stream, Serialization, Reader/Writers)
- Multi-threading (Thread vs Runnable, thread methods, Thread group, Synchronization, Deadlock)
- JVM Architecture, Reflection, Annotation, Garbage Collection

Reference Books

- Core Java Volume 1 & 2 - Horstmann
- Java Complete Reference - Herbert Schildt
- Java 8 Certification - Khalid Mughal
- Java Certification - Kathy Sierra

Java History

- Java goes back to 1991, when a group of Sun engineers, led by Patrick Naughton and James Gosling, wanted to design a small computer language that could be used for consumer devices like cable TV switchboxes.
- Since these devices do not have a lot of power or memory, the language had to be small and generate very tight code. Also, as different manufacturers may choose different CPU's, it was important that the language not be tied to any single architecture.
- The project was code-named "Green".

- The requirements for small, tight, and platform-neutral code led the team to design a portable language that generated intermediate code for a virtual machine.
- The Sun people came from a UNIX (Solaris) background, so they based their language on C++.
- Gosling decided to call his language "Oak". However, Oak was the name of an existing computer language, so they changed the name to Java.
- In 1992, the Green project delivered its first product, called "*7" - a smart remote control.
- Unfortunately, Sun was not interested in producing this, and the Green people had to find other ways to market their technology. However, none of the standard consumer electronics companies were interested either.
- The Green team spent all of 1993 and half of 1994 looking for sponsors to buy its technology.
- Meanwhile, the World Wide Web (WWW) was growing bigger and bigger. The key to the WWW was the browser translating hypertext pages to the screen.
- In 1994, most people were using Mosaic, a noncommercial web browser by University of Illinois.
- The Java language developers developed a cool browser - HotJava browser. This was client/server architecture-neutral application that was real-time and reliable.
- The developers made the browser capable of executing Java code inside web pages - called Applets. This POC was demonstrated at SunWorld on 23-May-1995.

Java versions

- JDK Beta - 1995
- **JDK 1.0 - January 23, 1996**
- JDK 1.1 - February 19, 1997
- **J2SE 1.2 - December 8, 1998**
    - Java collections
- J2SE 1.3 - May 8, 2000
- J2SE 1.4 - February 6, 2002
- **J2SE 5.0 - September 30, 2004**
    - enum
    - Generics
    - Annotations
- Java SE 6 - December 11, 2006
- Java SE 7 - July 28, 2011
- **Java SE 8 (LTS) - March 18, 2014**
    - Functional programming: Streams, Lambda expressions
- Java SE 9 - September 21, 2017
- Java SE 10 - March 20, 2018
- **Java SE 11 (LTS) - September 25, 2018**
- Java SE 12 - March 19, 2019
- Java SE 13 - September 17, 2019
- Java SE 14 - March 17, 2020
- Java SE 15 - September 15, 2020
- Java SE 16 - March 16, 2021
- **Java SE 17 (LTS) - September 14, 2021**
    - Jakarta SE 17
- Java SE 18 - March 22, 2022
- Java SE 19 - September 20, 2022

- Java SE 20 - March 21, 2023

Java platforms

- Java is not specific to any processor or operating system as Java platforms have been implemented for a wide variety of hardware and operating systems with a view to enable Java programs to run identically on all of them. Different platforms target different classes of device and application domains:
- **Java Card**: A technology that allows small Java-based applications to be run securely on smart cards and similar small-memory devices.
- **Java ME (Micro Edition)**: Specifies several different sets of libraries (known as profiles) for devices with limited storage, display, and power capacities. It is often used to develop applications for mobile devices, PDAs, TV set-top boxes, and printers.
- **Java SE (Standard Edition)**: Java Platform, Standard Edition or Java SE is a widely used platform for development and deployment of portable code for desktop environments
- **Java EE (Enterprise Edition)**: Java Platform, Enterprise Edition or Java EE is a widely used enterprise computing platform. The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications.

Object Oriented

- Basic principles of Object-oriented Language
  - class
  - object
- class
  - User defined data type (similar to struct in C)
  - Has fields (data members) and methods (member functions)
    - static members: Accessed using class name directly
    - non-static members: Accessed using object
  - Defines the structure/blueprint of the created object/instance
  - Logical entity
- object
  - Instance of the class
  - One class can have multiple objects
  - Physical entity (occupies memory)

Hello World - Code, Compilation and Execution

- Code

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

- Explanation - main():
  - In Java, each variable/method must be in some class.
  - JVM calls main() method without creating object of the class, so method must be static.
  - main() doesn't return any value to JVM, so return type is void.
  - main() takes command line arguments - String args[]
  - main() should be callable from outside the class directly - public access.
- Explanation - System.out.println():
  - System is predefined Java class (java.lang.System).
  - out is public static field of the System class --> System.out.
  - out is object of PrintStream class (java.io.PrintStream).
  - println() is public non-static method of PrintStream class --> System.out.println("...");
- Compilation and Execution (in same directory)
  - terminal> javac Program.java
  - terminal> java Program

Entry point method

- main() is considered as entry poit method in java.

```java
public static void main(String[] args) {
    // code
}
```

- JVM invokes main method.
- Can be overloaded.
- Can write one entry-point in each Java class.

java.lang.System class

```
```Java
public final class System
{
    //Fields
    public static final InputStream in;     // stdin
    public static final PrintStream out;    // stdout
    public static final PrintStream err;    // stderr
    //Methods
    public static Console console();
    public static void exit(int status);
    public static void gc();
    // ...
}
```
```

System.out.println()

- System: Final class declared in java.lang package and java.lang package is declared in rt.jar file.
- out: Object of PrintStream class declared as public static final field in System class.
- println(): Non-static method of PrintStream class.

C/C++ Program Compilation and Execution

- Main.cpp --> Compiler --> Main.obj --> Linker --> Main.exe
    - Main.cpp - Source code
    - Main.obj - Object code
    - Main.exe - Program = Executable code (contains machine level code)
- terminal> ./Main.exe
- Operating system creates a process to execute the program.
- Process sections
    - Text:
    - Data:
    - Rodata:
    - Stack:
    - Heap

Java Program Compilation and Execution

- Main.java --> Compiler --> Main.class --> JVM
    - Main.java --> Source code
    - Main.class --> Byte code (Intermediate Language code)
- terminal> javac Main.java
- terminal> java Main

JDK vs JRE vs JVM

- SDK is Software Development Kit required to develop application.
- SDK = Software Development Tools + Libraries + Runtime environment + Documentation + IDE.
    - Software Development Tools = Compiler, Debugger, etc.
    - Libraries = Set of functions/classes.
- JDK is Java platform SDK. It is a software development environment used for developing Java applications.
- JDK = Java Development Tools + JRE + Java docs.
    - Required to develop Java applications.
- JRE = Java API (Java class libraries) + Java Virtual Machine (JVM).
    - All core java fundamental classes are part of rt.jar file.
    - Required to develop and execute Java applications.

Hello World - Variations

- In STS Eclipse, classes are written under "src" directory. They are auto-compiled and generated .class files are placed under "bin" directory.
- One Java project can have multiple .java files. Each file can have main() method which can be executed separately.
- The main() method must be public static void. Missing any of them raise compiler error.

- The entry-point method must be main(String[] args). Otherwise, raise runtime error - main() method not found.
- The main() method can be overloaded i.e. method with same name but different parameters (in same class).
- If a .java file contains multiple classes, for each class a separate .class file is created.
- Name of (non-public) Java class may be different than the file name. The name of generated .class file is same as class name.
- Name of public class in Java file must be same as file-name. One Java file can have only one public class.

PATH vs CLASSPATH

- Enviorment variables: Contains important information about the system e.g. OS, CPU, PATH, USER, etc.
- PATH: Contains set of directories separated by ; (Windows) or : (Linux).
    - When any program (executable file) is executed without its full path (on terminal/Run), then OS search it in all directories given in PATH variable.
    - terminal> mspaint.exe
    - terminal> notepad.exe
    - terminal> taskmgr.exe
    - terminal> java.exe -version
    - terminal> javac.exe -version
    - To display PATH variable
        - Windows cmd> set PATH
        - Linux terminal> echo $PATH
    - PATH variable can be modified using "set" command (Windows) or "export" command (Linux).
    - PATH variable can be modified permanently in Windows System settings or Linux ~/.bashrc.
- CLASSPATH: Contains set of directories separated by ; (Windows) or : (Linux).
    - Java's environment variable by which one can inform Java compiler, application launcher, JVM and other Java tools about the directories in which Java classes/packages are kept.
    - CLASSPATH variable can be modified using "set" command (Windows) or "export" command (Linux).
        - Windows cmd> set CLASSPATH=\path\to\set;%CLASSPATH%
        - Linux terminal> export CLASSPATH=/path/to/set:$CLASSPATH
    - To display CLASSPATH variable
        - Windows cmd> set CLASSPATH
        - Linux terminal> echo $CLASSPATH
- Compilation and Execution (source code in "src" directory and .class file in "bin" directory)
    - terminal> cd \path\of\src directory
    - terminal> javac -d ..\bin Program.java
    - terminal> set CLASSPATH=..\bin
    - terminal> java Program

Console Input/Output

- Java has several ways to take input and print output. Most popular ways in Java 8 are given below:
- Using java.util.Scanner and System.out

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter name: ");
String name = sc.nextLine();
System.out.print("Enter age: ");
int age = sc.nextInt();
System.out.println("Name: " + name + ", Age: " + age);
System.out.printf("Name: %s, Age: %s\n", name, age);
```

- Using java.io.Console

```
Console console = System.console();
String email = console.readLine("Enter Email: ");
char[] passwd = console.readPassword("Enter Password: ");
console.printf("Email: %s\n", email);
```

Language Fundamentals

Naming conventions

- Names for variables, methods, and types should follow Java naming convention.
- Camel notation for variables, methods, and parameters.
  - First letter each word except first word should be capital.
  - For example:

```
public double calculateTotalSalary(double basicSalary, double
incentives) {
    double totalSalary = basicSalary + incentives;
    return totalSalary;
}
```

- Pascal notation for type names (i.e. class, interface, enum)
  - First letter each word should be capital.
  - For example:

```
class CompanyEmployeeManagement {
    // ...
}
```

- Package names must be in lower case only.
  - For example: javax.servlet.http;
- Constant fields must be in upper case only.
  - For example:

```java
final double PI = 3.14;
final int WEEKDAYS = 7;
final String COMPANY_NAME = "Sunbeam Infotech";
```

Keywords

- Keywords are the words whose meaning is already known to Java compiler.
- These words are reserved i.e. cannot be used to declare variable, function or class.
- Java 8 Keywords
    1. abstract - Specifies that a class or method will be implemented later, in a subclass
    2. assert - Verifies the condition. Throws error if false.
    3. boolean- A data type that can hold true and false values only
    4. break - A control statement for breaking out of loops.
    5. byte - A data type that can hold 8-bit data values
    6. case - Used in switch statements to mark blocks of text
    7. catch - Catches exceptions generated by try statements
    8. char - A data type that can hold unsigned 16-bit Unicode characters
    9. class - Declares a new class
    10. continue - Sends control back outside a loop
    11. default - Specifies the default block of code in a switch statement
    12. do - Starts a do-while loop
    13. double - A data type that can hold 64-bit floating-point numbers
    14. else - Indicates alternative branches in an if statement
    15. enum - A Java keyword is used to declare an enumerated type. Enumerations extend the base class.
    16. extends - Indicates that a class is derived from another class or interface
    17. final - Indicates that a variable holds a constant value or that a method will not be overridden
    18. finally - Indicates a block of code in a try-catch structure that will always be executed
    19. float - A data type that holds a 32-bit floating-point number
    20. for - Used to start a for loop
    21. if - Tests a true/false expression and branches accordingly
    22. implements - Specifies that a class implements an interface
    23. import - References other classes
    24. instanceof - Indicates whether an object is an instance of a specific class or implements an interface
    25. int - A data type that can hold a 32-bit signed integer
    26. interface- Declares an interface
    27. long - A data type that holds a 64-bit integer
    28. native - Specifies that a method is implemented with native (platform-specific) code
    29. new - Creates new objects
    30. null - This indicates that a reference does not refer to anything
    31. package - Declares a Java package
    32. private - An access specifier indicating that a method or variable may be accessed only in the class it’s declared in

33. protected - An access specifier indicating that a method or variable may only be accessed in the class it's declared in (or a subclass of the class it's declared in or other classes in the same package)
34. public - An access specifier used for classes, interfaces, methods, and variables indicating that an item is accessible throughout the application (or where the class that defines it is accessible)
35. return - Sends control and possibly a return value back from a called method
36. short - A data type that can hold a 16-bit integer
37. static - Indicates that a variable or method is a class method (rather than being limited to one particular object)
38. strictfp - A Java keyword is used to restrict the precision and rounding of floating-point calculations to ensure portability.
39. super - Refers to a class's base class (used in a method or class constructor)
40. switch - A statement that executes code based on a test value
41. synchronized - Specifies critical sections or methods in multithreaded code
42. this - Refers to the current object in a method or constructor
43. throw - Creates an exception
44. throws - Indicates what exceptions may be thrown by a method
45. transient - Specifies that a variable is not part of an object's persistent state
46. try - Starts a block of code that will be tested for exceptions
47. void - Specifies that a method does not have a return value
48. volatile - This indicates that a variable may change asynchronously
49. while - Starts a while loop
50. goto, const - Unused keywords (Reserved words)
51. true, false, null - Literals (Reserved words)

Data types

- Data type describes:
  - Memory is required to store the data
  - Kind of data memory holds
  - Operations to perform on the data
- Java is strictly type checked language.
- In java, data types are classified as:
  - Primitive types or Value types
  - Non-primitive types or Reference types

```
Data types
    |- Primitive types (Value types)
    |      |- Boolean: boolean
    |      |- Character: char
    |      |- Integral: byte, short, int, long
    |      |- Floating-point: float, double
    |
    |- Non-Primitive types (Reference types)
           |- class
           |- interface
           |- enum
           |- Array
```

1. boolean (size is not specified)
2. byte (size is 1 byte)
3. char (size is 2 bytes)
4. short (size is 2 bytes)
5. int (size is 4 bytes)
6. float (size is 4 bytes)
7. double (size is 8 bytes)
8. long (size is 8 bytes)

- primitive types(boolean, byte, char, short, int ,float, double, long) are not classes in Java.

```
Stack<int> s1 =  new Stack<int>( ); //Not OK
Stack<Integer> s1 =  new Stack<Integer>( ); //OK
```

Datatype	Detail	Default	Memory needed (size)	Examples	Range of Values
boolean	It can have value true or false, used for condition and as a flag.	false	1 bit	true, false	true or false
byte	Set of 8 bits data	0	8 bits	NA	-128 to 127
char	Used to represent chars	\u0000	16 bits	"a", "b", "c", "A" and etc.	Represents 0-256 ASCII chars
short	Short integer	0	16 bits	NA	-32768-32768
int	integer	0	32 bits	0, 1, 2, 3, -1, -2, -3	-2147483648 to 2147483647-
long	Long integer	0	64 bits	1L, 2L, 3L, -1L, -2L, -3L	-9223372036854775807 to 9223372036854775807
float	IEEE 754 floats	0.0	32 bits	1.23f, -1.23f	Upto 7 decimal
double	IEEE 754 floats	0.0	64 bits	1.23d, -1.23d	Upto 16 decimal

- Widening: We can convert state of object of narrower type into wider type. it is called as "widening".

```
int num1 = 10;
double num2 = num1; //widening
```

- Narrowing: We can convert state of object of wider type into narrower type. It is called "narrowing".

```
double num1 = 10.5;
int num2 = (int) num1; //narrowing
```

- Rules of conversion
  - source and destination must be compatible i.e. destination data type must be able to store larger/equal magnitude of values than that of source data type.
  - Rule 1: Arithmetic operation involving byte, short automatically promoted to int.
  - Rule 2: Arithmetic operation involving int and long promoted to long.
  - Rule 3: Arithmetic operation involving float and long promoted to float.
  - Rule 4: Arithmetic operation involving double and any other type promoted to double.
- Type Conversions

Literals

- Six types of Literals:
  - Integral Literals
  - Floating-point Literals
  - Char Literals
  - String Literals
  - Boolean Literals
  - null Literal

Integral Literals

- Decimal: It has a base of ten, and digits from 0 to 9.
- Octal: It has base eight and allows digits from 0 to 7. Has a prefix 0.
- Hexadecimal: It has base sixteen and allows digits from 0 to 9 and A to F. Has a prefix 0x.
- Binary: It has base 2 and allows digits 0 and 1.
- For example:

```
int x = 65; // decimal const don't need prefix
int y = 0101; // octal values start from 0
int z = 0x41; // hexadecimal values start from 0x
int w = 0b01000001; // binary values start with 0b
```

- Literals may have suffix like U, L.
  - L -- represents long value.

```
long x = 123L; // long const assigned to long variable
long y = 123; // int const assigned to long variable -- widening
```

Floating-Point Literals

- Expressed using decimal fractions or exponential (e) notation.

- Single precision (4 bytes) floating-point number. Suffix f or F.
- Double precision (8 bytes) floating-point number. Suffix d or D.
- For example:

```
float x = 123.456f;
float y = 1.23456e+2;   // 1.23456 x 10^2 = 123.456
double z = 3.142857d;
```

Char Literals

- Each char is internally represented as integer number - ASCII/Unicode value.
- Java follows Unicode char encoding scheme to support multiple langauges.
- For example:

```
char x = 'A';       // char representation
char y = '\101';    // octal value
char z = '\u0041';  // unicode value in hex
char w = 65;        // unicode value in dec as int
```

- There are few special char literals referred as escape sequences.
  - \n -- newline char -- takes cursor to next line
  - \r -- carriage return -- takes cursor to start of current line
  - \t -- tab (group of 8 spaces)
  - \b -- backspace -- takes cursor one position back (on same line)
  - ' -- single quote
  - " -- double quote
  - \ -- prints single \
  - \0 -- ascii/unicode value 0 -- null character

String Literals

- A sequence of zero or more unicode characters in double quotes.
- For example:

```
String s1 = "Sunbeam";
```

Boolean Literals

- Boolean literals allow only two values i.e. true and false. Not compatible with 1 and 0.
- For example:

```
boolean b = true;
boolean d = false;
```

Null Literal

- "null" represents nothing/no value.
- Used with reference/non-primitive types.

```
String s = null;
Object o = null;
```

Variables

- A variable is a container which holds a value. It represents a memory location.
- A variable is declared with data type and initialized with another variable or literal.
- In Java, variable can be
  - Local: Within a method -- Created on stack.
  - Non-static/Instance field: Within a class - Accessed using object.
  - Static field: Within a class - Accessed using class-name.

Operators

- Java divides the operators into the following catgories:
  - Arithmetic operators: +, -, *, /, %
  - Assignment operators: =, +=, -=, etc.
  - Comparison operators: ==, !=, <, >, <=, >=, instanceof
  - Logical operators: &&, ||, !
    - Combine the conditions (boolean - true/false)
  - Bitwise operators: &, |, ^, ~, <<, >>, >>>
  - Misc operators: ternary ?:, dot .
    - Dot operator: ClassName.member, objName.member.

- Operator precedence and associativity

Operator	Description	Associativity
++ --	unary postfix increment unary postfix decrement	right to left
++ -- + - ! ~ (*type*)	unary prefix increment unary prefix decrement unary plus unary minus unary logical negation unary bitwise complement unary cast	right to left
* / %	multiplication division remainder	left to right
+ -	addition or string concatenation subtraction	left to right
<< >>>>>>>	left shift signed right shift unsigned right shift	left to right
< <= > >= instanceof	less than less than or equal to greater than greater than or equal to type comparison	left to right
== !=	is equal to is not equal to	left to right
&	bitwise AND boolean logical AND	left to right

jdk-8.0.372.7-hotspot
This PC › Local Disk (C:) › Program Files › Eclipse Adoptium › jdk-8.0.372.7-hotspot
bin
include
jre
lib
sample
ASSEMBLY_EXCEPTION
LICENSE
NOTICE
release
src.zip
THIRD_PARTY_README
Java binaries -- App launcher (java.exe), Java compiler (javac.exe), Java debugger, ...
JVM (jvm.dll) + Tools to execute Java i.e. App launcher (java.exe)
Java implementation libs (internal JVM impl) + Java lib (rt.jar)
Header files used for JNI programming
Java source code.
Java example codes.
Select a file to preview.
11 items

Software
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Select all
Select none
Invert selection
Select
This PC › Local Disk (C:) › Nilesh › onlinecourses › CJ-H-02 › private › Software ›
Search Software
$WinREAgent
.Trash-1000
DRIVERS
Intel
MahaSecure
MinGW
MSOCache
Nilesh
day6
files
internship
mar-22
mar-23
onlinecourses
aspnet_mvc_01
bdh-01
bdi-01
cj-01
cj-10
cj-12
CJ-H-02
.git
private
Books
Software
sts-4.19.0.RELEASE
Name
Date modified
Type
Size
sts-4.19.0.RELEASE
1/29/2024 10:18 AM
File folder
jdk-11.0.20_doc-all.zip
8/2/2023 10:42 PM
Compressed (zipp...
52.51
OpenJDK11U-jdk_x64_windows_hotspot_...
1/29/2024 7:58 AM
Windows Installer ...
171.20
spring-tool-suite-4-4.19.0.RELEASE-e4.28...
8/2/2023 10:29 PM
JAR File
592.95
Double click to extract Eclipse STS.
Select a file to preview.
4 items
Search
10:19 AM

sts-4.19.0.RELEASE
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Select all
Select none
Invert selection
Select
This PC › Local Disk (C:) › Nilesh › onlinecourses › CJ-H-02 › private › Software › sts-4.19.0.RELEASE ›
Search sts-4.19.0.RELEASE
$WinREAgent
.Trash-1000
DRIVERS
Intel
MahaSecure
MinGW
MSOCache
Nilesh
day6
files
internship
mar-22
mar-23
onlinecourses
aspnet_mvc_01
bdh-01
bdi-01
cj-01
cj-10
cj-12
CJ-H-02
.git
private
Books
Software
sts-4.19.0.RELEASE
Name
Date modified
Type
Size
configuration 1/29/2024 10:18 AM File folder
dropins 1/29/2024 10:18 AM File folder
features 1/29/2024 10:18 AM File folder
p2 1/29/2024 10:18 AM File folder
plugins 1/29/2024 10:18 AM File folder
readme 1/29/2024 10:18 AM File folder
.eclipseproduct 1/29/2024 10:18 AM ECLIPSEPRODUCT ...
artifacts.xml 1/29/2024 10:18 AM XML Source File 15
license.txt 1/29/2024 10:18 AM Text Document 1
open-source-licenses.txt 1/29/2024 10:18 AM Text Document 75
SpringToolSuite4.exe 1/29/2024 10:18 AM Application 52
SpringToolSuite4.ini 1/29/2024 10:18 AM Configuration setti...
SpringToolSuite4c.exe 1/29/2024 10:18 AM Application 23
Select a file to preview.
Double click to start Eclipse.
13 items
Search
10:20 AM

sts-4.19.0.RELEASE
Manage
File
Home
Share
View
Application Tools
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Select all
Select none
Invert selection
Select
This PC › Local Disk (C:) › Nilesh › onlinecourses › CJ-H-02 › private › Software › sts-4.19.0.RELEASE ›
Search sts-4.19.0.RELEASE
.Trash-1000
DRIVERS
Intel
MahaSecure
MinGW
MSOCache
Nilesh
day6
files
internship
mar-22
mar-23
onlinecourses
aspnet_mvc_01
bdh-01
bdi-01
cj-01
cj-10
cj-12
CJ-H-02
.git
day01
private
Books
Software
sts-4.19.0.RELEASE
Name
Date modified
Type
Size
configuration
dropins
features
p2
plugins
readme
.eclipseproduct
artifacts.xml
license.txt
open-source-licenses.txt
SpringToolSuite4.exe
SpringToolSuite4.ini
SpringToolSuite4c.exe
Spring Tool Suite 4 Launcher
Select a directory as workspace
Spring Tool Suite 4 uses the workspace directory to store its preferences and development artifacts.
C:\Nilesh\onlinecourses\CJ-H-02\day01
Browse...
Use this as the default and do not ask again
Recent Workspaces
Launch
Cancel
No preview available.
13 items
1 item selected 520 KB
Search
10:21 AM

day01 - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
New Alt+Shift+N >
Open File...
Open Projects from File System...
Recent Files
Close Editor Ctrl+W
Close All Editors Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
Switch Workspace
Restart
Exit
Java Project
Maven Project
Spring
Create a Java project
Import Spring Getting Started Content
Project...
Package
Class
Interface
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Other... Ctrl+N
Open a file or drop files here to open them.
Find Actions Ctrl+3
Show Key Assist Ctrl+Shift+L
New Ctrl+N
Open Type Ctrl+Shift+T
Problems Javadoc Declaration
0 items
Description Resource Path Location Type
Search
10:24 AM

day01 - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Windo

Package Explorer

There are no projects in your workspace.
To add a project:

- Create a Java project
- Create a Maven project
- Create new Spring Starter Project
- Import Spring Getting Started Content
- Create a project...
- Import projects...

Problems

0 items

Description

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: first

☑ Use default location

Location: C:\Nilesh\onlinecourses\CJ-H-02\day01\first Browse...

JRE

◉ Use an execution environment JRE: JavaSE-1.8

○ Use a project specific JRE: jre

○ Use default JRE 'jre' and workspace compiler preferences Configure JREs...

Project layout

○ Use project folder as root for sources and class files

◉ Create separate folders for sources and class files Configure default...

Working sets

☐ Add project to working sets New...

Working sets: Select...

Module

☐ Create module-info.java file

Module name:

☑ Generate comments

ⓘ The default compiler compliance level for the current workspace is 17. The new project will use a project specific compiler

< Back Next > Finish Cancel

Search 10:25 AM

day01 - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Windo
Package Explorer
> first [CJ-H-02 main 11]
New Java Project
Java Settings
Define the Java build settings.
Source
Projects
Libraries
Order and Export
Module Dependencies
> first [CJ-H-02 main 11]
src
Details
Create new source folder: use this if you want to add a new source folder to your project.
Link additional source: use this if you have a folder in the file system that should be used as additional source folder.
Add project 'first' to build path: Add the project to the build path if the project is the root of packages and source files. Entries on the build path are visible to the compiler and used for building.
Allow output folders for source folders
Default output folder:
first/bin
Browse...
< Back
Next >
Finish
Cancel
Problems
0 items
Description
Search
10:25 AM

day01 - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
first [CJ-H-02 main 11]
JRE System Library [JavaSE-1.8]
src
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Source
Refresh F5
Assign Working Sets...
Run As
Debug As
Profile As
Restore from Local History...
Team
Compare With
Replace With
Configure
Properties Alt+Enter
Java Project
Maven Project
Spring Starter Project
Import Spring Getting Started Content
Project...
Package
Class
Interface
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Other... Ctrl+N
Open a file or drop files here to open them.
Find Actions Ctrl+3
Show Key Assist Ctrl+Shift+L
New Ctrl+N
Open Type Ctrl+Shift+T
Javadoc Declaration
Description Resource Path Location Type
src - first
Search
10:26 AM

day01 - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

> first [CJ-H-02 main 11]
  - JRE System Library [JavaSE-1.8]
  - src

New Java Class

Java Class

The use of the default package is discouraged.

Source folder: first/src Browse...

Package: (default) Browse...

Enclosing type: Browse...

Name: Program

Modifiers: public package private protected

abstract final static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

- public static void main(String[] args)
- Constructors from superclass
- Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

- Generate comments

Finish Cancel

Problems Javadoc Declaration

0 items

Description	Resource	Path	Location	Type

src - first

Search 10:27 AM

day01 - first/src/Program.java - Spring Tool Suite 4

File Edit Source Refactor Source Navigate Search Project Run Window Help

Package Explorer

- first [CJ-H-02 main 11]
  - JRE System Library [JavaSE-1.8]
  - src
    - (default package)
      - Program.java

Program.java

```
1
2 public class Program {
3     public static void main(String args[]) {
4         System.out.println("Hello, World!!");
5     }
6 }
7
```

Right click

Undo	Ctrl+Z
Revert File	
Save	Ctrl+S
Open Declaration	F3
Open Type Hierarchy	
Open Call Hierarchy	Ctrl+Alt+H
Show in Breadcrumb	Alt+Shift+B
Quick Outline	Ctrl+O
Quick Type Hierarchy	Ctrl+T
Open With	›
Show In	Alt+Shift+W ›
Cut	Ctrl+X
Copy	Ctrl+C
Copy Qualified Name	
Paste	Ctrl+V
Raw Paste	
Quick Fix	Ctrl+1
Source	Alt+Shift+S ›
Refactor	Alt+Shift+T ›
Local History	›
References	›
Declarations	›
Run As	›
Debug As	›
Profile As	›
Team	›
Compare With	›
Replace With	›
Preferences...	

1 Run on Server	Alt+Shift+X, R
2 Java Application	Alt+Shift+X, J
3 Java Application In Container	
Run Configurations...	

Problems × Javadoc Declara

0 items

Description	Location	Type

Writable | Smart Insert | 7 : 1 : 117

Search 10:28 AM

day01
File Home Share View
Pin to Quick access Copy Paste Cut Copy path Paste shortcut
Clipboard
Move to Copy to Delete Rename
Organize
New folder New item Easy access
Properties Open Edit History
Select all Select none Invert selection
This PC > Local Disk (C:) > Nilesh > onlinecourses > CJ-
.Trash-1000
DRIVERS
Intel
MahaSecure
MinGW
MSOCache
Nilesh
day6
files
internship
mar-22
mar-23
onlinecourses
aspnet_mvc_01
bdh-01
bdi-01
cj-01
cj-10
cj-12
CJ-H-02
.git
day01
private
Books
Software
sts-4.19.0.RELEASE
Name
.metadata
first
2 items
*Untitled - Notepad
File Edit Format View Help
public class Program {
public static void main(String args[]) {
System.out.println("Hello, World!!!");
}
}
Save As
onlinecourses > CJ-H-02 > day01
Search day01
Organize New folder
Name Date modified Type
.metadata 1/29/2024 10:21 AM File folder
first 1/29/2024 10:25 AM File folder
File name: Program.java
Save as type: All Files (*.*)
Hide Folders
Encoding: UTF-8
Save
Cancel
Search
10:34 AM

day01
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Select all
Select none
Invert selection
Select
This PC › Local Disk (C:) › Nilesh › onlinecourses › CJ-H-02 › day01
Search day01
.Trash-1000
DRIVERS
Intel
MahaSecure
MinGW
MSOCache
Nilesh
day6
files
internship
mar-22
mar-23
onlinecourses
aspnet_mvc_01
bdh-01
bdi-01
cj-01
cj-10
cj-12
CJ-H-02
.git
day01
private
Books
Software
sts-4.19.0.RELEASE
Name
Date modified
Type
Size
.metadata
1/29/2024 10:21 AM
File folder
first
1/29/2024 10:25 AM
File folder
Program.java
1/29/2024 10:34 AM
Java Source File
View
Sort by
Group by
Refresh
Customize this folder...
Paste
Paste shortcut
Undo Rename
Ctrl+Z
Open in Terminal
Open with Visual Studio
Git GUI Here
Git Bash Here
Open with Code
Give access to
New
Properties
Select a file to preview.
3 items
Search
10:34 AM

day01
File
Home
Share
View
This PC > Local Disk (C:) > Nilesh > onlinecourses > CJ-H-02 > day01 >
Search day01
Name
Date modified
Type
Size
.metadata 1/29/2024 10:21 AM File folder
first 1/29/2024 10:25 AM File folder
Program.class 1/29/2024 10:35 AM CLASS File
Program.java 1/29/2024 10:34 AM Java Source File
javac --> Java compiler
java --> Java application launcher
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Nilesh\onlinecourses\CJ-H-02\day01> javac Program.java
PS C:\Nilesh\onlinecourses\CJ-H-02\day01>
PS C:\Nilesh\onlinecourses\CJ-H-02\day01> java Program
Hello, World!!!
PS C:\Nilesh\onlinecourses\CJ-H-02\day01>
Compiles Java source code into Java byte code (.class file)
Executes Java program/class.
4 items

day01 - first/src/Program.java - Spring Tool Suite 4

File Edit Source Refactor Source Navigate Search Project Run Window Help

In Java, every data/function must be inside some class.

Program.java

class is accessible (outside the package).

```
1
2 public class Program {
3     public static void main(String args[]) {
4         System.out.println("Hello, World!!");
5     }
6 }
7
```

entry-point of the java appln i.e. JVM begin app execution from this method.

method is accessible outside the class.

System - is a class java.lang.System

out - static field in System class. out is object of PrintStream class.

System.out -- to display output on terminal/console (stdout).

println() -- method in PrintStream class to display message/values.

void - return type - method doesn't return any value.

String[] args - cmd line args i.e. values given on cmdline while executing program (if any).

e.g. > java Program arg1 arg2 arg3

static - belongs to the class (not obj) to be called with class name (without object of the class).

JVM calls main() method without creating object of the class.

Java is Object Oriented.

==========

Basic OOP concepts

* class
* object

class:

- user defined data type (like struct in C)
- fields + methods
- logical entity = blueptint of the object

object:

- instance/var of class
- physical entity (allocated on heap)

Writable | Smart Insert | 1 : 1 : 0

Search 11:45 AM

day01 - second/src/Program.java - Spring Tool Suite 4

File Edit Source Refactor Source Navigate Search Project Run Window Help

Program.java

```java

/*
 * Write a Program to calculate area of rectangle.
 * */

public class Program {
    public static void main(String[] args) {
        // variable declarations
        double length = 10.0, breadth = 5.5, area;
        // area calculation
        area = length * breadth;
        // print result
        System.out.println("Area = " + area);
        System.out.printf("Area = %f\n", area);
    }
}

```

Java is strongly typed language.

Basic data types: int, float, double, char, ...

- int -- whole numbers
- double -- decimal/fractional numbers
- char -- characters

length	breadth	area	<-- on stack
10.0	5.5	55.0	

Can print values just by string concatenation

System.out.println("len:"+length+"br:"+breadth+ "area:"+area);

printf() can print values using format specifier

e.g. %d for int, %f for double, %s for String

Writable | Smart Insert | 12 : 24 : 276

Search 12:21 PM

day01 - second/src/Program.java - Spring Tool Suite 4

File Edit Source Refactor Source Navigate Search Project Run Window Help

Package Explorer

- first [CJ-H-02 main ↑1]
  - JRE System Library [JavaSE-1.8]
  - src
    - (default package)
      - Program.java
- second [CJ-H-02 main ↑1]
  - JRE System Library [JavaSE-1.8]
  - src
    - (default package)
      - Hello.java
      - Program.java

Program.java

```java
/*
 * Write a Program to calculate area of
 * */

public class Program {
    public static void main(String[] arg
        // variable declarations
        double length = 10.0, breadth = 
        // area calculation
        area = length * breadth;
        // print result
        System.out.println("Area = " + a
        System.out.printf("Area = %f\n",
    }
}
```

Hello.java

```java


public class Hello {
    public static void main(String[] arg
        System.out.println("Hello, Java!
    }
}
```

In one class, there can be only one main() method as public static void main(String[] args).

Problems Javadoc Declaration Console

<terminated> Program (1) [Java Application] C:\Nilesh\onlinecourses\CJ-H-02\private\Software\sts-4.19.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230425-1502\jre\bin\javaw.exe (Jan 29, 2024, 12:33:11 PM – 1

```
Area = 55.0
Area = 55.000000
```

One Java project can have multiple classes.
Each class can have separate main() method that can be executed
- right click on class - Run as - Java Application.

Writable | Smart Insert | 7 : 25 : 112

Search 12:35 PM

A .java file can have one or more java classes. When compiled one .class is created for each java class.

This PC > Local Disk (C:) > Nilesh > onlinecourses > CJ-H-02 > day01

Name	Date modified	Type
.metadata	1/29/2024 10:21 AM	File folder
first	1/29/2024 10:25 AM	File folder
second	1/29/2024 11:38 AM	File folder
MyClass.class	1/29/2024 12:43 PM	CLASS File
MyProgram.class	1/29/2024 12:43 PM	CLASS File
MyProgram.java	1/29/2024 12:40 PM	Java Source
Program.java	1/29/2024 10:34 AM	Java Source
YourClass.class	1/29/2024 12:43 PM	CLASS File

```
PS C:\Nilesh\onlinecourses\CJ-H-02\day01> javac MyProgram.ja
va
PS C:\Nilesh\onlinecourses\CJ-H-02\day01> java MyProgram
MyProgram.main()
PS C:\Nilesh\onlinecourses\CJ-H-02\day01> java MyClass
MyClass.main()
PS C:\Nilesh\onlinecourses\CJ-H-02\day01> java YourClass
YourClass.main()
PS C:\Nilesh\onlinecourses\CJ-H-02\day01>
```

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

class YourClass {
  public static void main(String[] args) {
     System.out.println("YourClass.main
()");
  }
}

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

There can be separate main() in each Java class.

It should be executed with separate "java" cmd.

day01
File Home Share View
Pin to Quick access Copy Paste Cut Copy path Paste shortcut Clipboard
Move to Copy to Delete Rename Organize
New folder New item Easy access New
Properties Open Edit History Open
Select all Select none Invert selection Select
This PC > Local Disk (C:) > Nilesh > onlinecourses > CJ-H-02 > day01
.Trash-1000
DRIVERS
Intel
MahaSecure
MinGW
MSOCache
Nilesh
day6
files
internship
Name Date modified Type Size
.metadata 1/29/2024 10:21 AM File folder
first 1/29/2024 10:25 AM File folder
second 1/29/2024 11:38 AM File folder
MyProgram.java 1/29/2024 12:48 PM Java Source File
Program.java 1/29/2024 10:34 AM Java Source File
One .java file can have multiple classes, but max one class can be public.
Windows PowerShell
PS C:\Nilesh\onlinecourses\CJ-H-02\day01> javac MyProgram.java
MyProgram.java:1: error: class MyClass is public, should be declared in a file named MyClass.java
public class MyClass {
^
MyProgram.java:7: error: class YourClass is public, should be declared in a file named YourClass.java
public class YourClass {
^
2 errors
PS C:\Nilesh\onlinecourses\CJ-H-02\day01>
MyProgram.java - Notepad
File Edit Format View Help
error
public class MyClass {
public static void main(String[] args) {
System.out.println("MyClass.main()");
}
}
error
public class YourClass {
public static void main(String[] args) {
System.out.println("YourClass.main()");
}
}
okay
public class MyProgram {
public static void main(String[] args) {
System.out.println("MyProgram.main()");
}
}
Also, name of public class must be same as name of .java file.
Multiple public classes should be written in multiple .java files.
Ln 1, Col 8 80% Windows (CRLF) UTF-8
Search
12:50 PM

day01

This PC > Local Disk (C:) > Nilesh > onlinecourses > CJ-H-02 > day01

Name	Date modified	Type	Size
.metadata	1/29/2024 10:21 AM	File folder	
first	1/29/2024 10:25 AM	File folder	
second	1/29/2024 11:38 AM	File folder	
MyClass.class	1/29/2024 12:55 PM	CLASS File	
MyProgram.class	1/29/2024 12:55 PM	CLASS File	
MyProgram.java	1/29/2024 12:54 PM	Java Source File	
Program.java	1/29/2024 10:34 AM	Java Source File	
YourClass.class	1/29/2024 12:55 PM	CLASS File	

Windows PowerShell

```
PS C:\Nilesh\onlinecourses\CJ-H-02\day01> javac MyProgram.java
PS C:\Nilesh\onlinecourses\CJ-H-02\day01>
```

no error.

MyProgram.java - Notepad

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

class YourClass {
  public static void main(String[] args) {
     System.out.println("YourClass.main()");
  }
}

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

only one public class = name same as file name.

Ln 13, Col 3 | 80% | Windows (CRLF) | UTF-8

```
PS C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01> javac Program.java
PS C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01>
PS C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01> java Program
Hello, Java!!
PS C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01>
PS C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01>
```

CLASSPATH variable stores set of directories separated by ; (on windows) or : (on Linux).
It helps Java tools (like compiler, app launcher, debugger, etc) and JVM to locate java classes and java packages.
These tools/JVM auto search java classes in all directories given in CLASSPATH variable.

By default CLASSPATH is not set. In this case, java classes/packages are located in current dir.

bin

This PC > Local Disk (C:) > Nilesh > onlinecourses > CJ-H-02 > day02 > demo01 > bin

Name	Date modified	Type	Size
Program.class	1/30/2024 10:18 AM	CLASS File	1 KB

Command Prompt

```
C:\Users\Nilesh>cd C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01\src

C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01\src>javac -d ..\bin Program.java

C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01\src>java Program
Error: Could not find or load main class Program
Caused by: java.lang.ClassNotFoundException: Program

C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01\src>set CLASSPATH
Environment variable CLASSPATH not defined

C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01\src>set CLASSPATH=../bin

C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01\src>set CLASSPATH
CLASSPATH=../bin

C:\Nilesh\onlinecourses\CJ-H-02\day02\demo01\src>java Program
Hello, Java!!
```

produce output .class file in given dir.

since CLASSPATH is not set, it will search Program class in current directory. Raise ClassNotFoundEx.

set the CLASSPATH to dir in which our .class file is stored.

now Java app launcher can auto find Program class and execute

1 item

When any source file is written/modified in src directory of the project, Eclipse build/compile is automatically and create .class file in "bin" directory.

day02 - demo02/src/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.
  - src
    - (default package)
      - Program01.java
      - Program02.java

Program01.java | Program02.java

```java
public class Program02 {
    // main() entry point -- called by JVM
    public static void main(String[] args) {
        System.out.println("main() entry-point function");
        Program02.main(22, 7);
        Program02.main();
    }
    // overloaded main() with two int args
    public static void main(int x, int y) {
        System.out.println("main() with two int arguments");
    }
    // overloaded main() with no args
    public static void main() {
        System.out.println("main() with no arguments");
    }
}
```

in Java, multiple methods in same scope can have same name but different arguments.
This is called as "method overload".

main() method can also be overloadec

However, JVM can only execute public static void main(String[] a);

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Jan 30, 2024, 10:32:17 AM – 10:32:17 AM) [pid: 10560]

```
main() entry-point function
main() with two int arguments
main() with no arguments
```

Writable | Smart Insert | 13 : 33 : 389

Search | 10:32 AM

day02 - demo02/src/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.
  - src
    - (default package)
      - Program01.java
      - Program02.java
      - Program03.java

Program01.java | Program02.java | Program03.java

```
1
2 public class Program03 {
3     public static void Main(String[] args) {
4         System.out.println("Hello, Java!");
5     }
6 }
7
```

In Java all method/var names are case sensitive.

Main() is not entry-point.

When executed from command line, will raise error i.e. main() not found.,

Problems | Javadoc | Declaration | Console

No consoles to display at this time.

Search 10:35 AM

day02 - demo03/src/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Ex... | Program01.java | Program02.java

```java
import java.util.Scanner;

public class Program02 {
    public static void main(String[] args) {
        // input user name, roll, and marks from user and display it back
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter roll: ");
        int roll = sc.nextInt();
        System.out.print("Enter name: ");
        String name = sc.nextLine();
        System.out.print("Enter marks: ");
        double marks = sc.nextDouble();

        System.out.println("Name: " + name);
        System.out.println("Roll: " + roll);
        System.out.println("Marks: " + marks);
    }
}
```

remains in input buffer.

7 +enter (\n)

Read until \n, but prev \n is already in input buffer, so further input not taken.
The "name" is empty String/.

Problems | Javadoc | Declaration | Console

<terminated> Program02 (1) [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Jan 30, 2024, 11:01:17 AM – 11:01:36 AM) [pid: 11596]

```
Enter roll: 7
Enter name: Enter marks: 99.9
Name:
Roll: 7
Marks: 99.9
```

Name was not input from user.

Search 11:02 AM

day02 - demo03/src/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Ex... | Program01.java | Program02.java

```java
import java.util.Scanner;

public class Program02 {
    public static void main(String[] args) {
        // input user name, roll, and marks from user and display it back
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter roll: ");
        int roll = sc.nextInt();
        System.out.print("Enter name: ");
        //String name = sc.nextLine();
        String name = sc.next();
        System.out.print("Enter marks: ");
        double marks = sc.nextDouble();

        System.out.println("Name: " + name);
        System.out.println("Roll: " + roll);
        System.out.println("Marks: " + marks);
    }
```

Doesn't skip due to \n,
But doesn't input multi-word string
(separated by space).

Problems | Javadoc | Declaration | Console

<terminated> Program02 (1) [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Jan 30, 2024, 11:06:19 AM – 11:06:44 AM) [pid: 13772]

```
Enter roll: 7
Enter name: JamesBond
Enter marks: 99.9
Name: JamesBond
Roll: 7
Marks: 99.9
```

Search 11:06 AM

day02 - demo03/src/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Ex... | Program01.java | Program02.java

```
1 import java.util.Scanner;
2
3 public class Program02 {
4     public static void main(String[] args) {
5         // input user name, roll, and marks from user and display it back
6         Scanner sc = new Scanner(System.in);
7         System.out.print("Enter roll: ");
8         int roll = sc.nextInt();    // user inputs 7 + \n. 7 is assigned to roll. \n remains in input buffer.
9         System.out.print("Enter name: ");
10        sc.nextLine();  // reads \n from input buffer and discard it (not assigned to any var).
11        String name = sc.nextLine(); // reads user input with spaces until \n e.g. James Bond
12        //String name = sc.next();
13        System.out.print("Enter marks: ");
14        double marks = sc.nextDouble(); // user inputs 99.9 + \n. 99.9 is assigned to marks. \n remains in in
15
16        System.out.println("Name: " + name);
17        System.out.println("Roll: " + roll);
18        System.out.println("Marks: " + marks);
```

Problems | Javadoc | Declaration | Console

<terminated> Program02 (1) [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Jan 30, 2024, 11:09:51 AM – 11:10:00 AM) [pid: 12364]

```
Enter roll: 7
Enter name: James Bond
Enter marks: 99.9
Name: James Bond
Roll: 7
Marks: 99.9
```

Search 11:10 AM

```
    * operations to perform on the data
* Java is strictly type checked language.
* In java, data types are classified as:
    * Primitive types or Value types
    * Non-primitive types or Reference types
    ```
    Data types
        |- Primitive types (Value types)
        | |- Boolean: boolean
        | |- Character: char
        | |- Integral: byte, short, int, long
        | |- Floating-point: float, double
        |
        |- Non-Primitive types (Reference types)
                |- class e.g. String, Scanner
                |- interface
                |- enum
                |- Array
    ```
* ![Data Types](https://tutorialshut.com/wp-content/uploads/2021/03/tabular-data.png)
* Widening: We can convert state of object of narrower type into wider type. it is called as "widening".
    ```Java
    int num1 = 10;
```

int a = 123;
double b = 3.14;

Actual objects always in Heap.

String str = "Java";
Scanner sc = new Scanner(System.in);

```

y02.md ×

- Rule 1: Arithmetic operation involving byte, short automatically promoted to int.
- Rule 2: Arithmetic operation involving int and long promoted to long.
- Rule 3: Arithmetic operation involving float and long promoted to float.
- Rule 4: Arithmetic operation involving double and any other type promoted to double.

smaller --> larger
a.k.a. widening (auto)
larger --> smaller
a.k.a. narrowing
(need casting)

These conversions (shown by arrow) are supported directly (i.e. no casting is required).

The reverse converions need explicit casting.

```
int a = 65;
short b = a; // error
short b = (short) a; // okay
short c = 12;
int d = c; // okay
```

Java Compilation

Program.java

```
class MyProgram {
  public static void
  main(String [] args) {
    =
  }
}
```

↓

Java Compiler

↓

MyProgram.class

↓

Java app launcher

Program.java

```
class A {
}
class B {
}
class C {
}
class Program {
  p.s.v. main (String[] args)
  {
    =
  }
}
```

→ Java compiler

↓

A.class

B.class

C.class

Program.class

↓

Java app launcher

Boxing / Unboxing

```
int i = 123;                          //boxing
Integer j = new Integer(i);
Integer k = i; //auto-boxing (5.0 +)
```

```
Integer x = new Integer(123);
int y = x.intValue(); //unboxing
int z = x; // auto-unboxing (5.0+)
```

Core Java

Java Buzzwords/Features

1. Simple
    - Java was designed to be easy for a professional programmer to learn and use effectively.
    - It's simple and easy to learn if you already know the basic concepts of Object Oriented Programming.
    - If you are an experienced C++ programmer, moving to Java will require very little effort. Because Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java.
    - Java has removed many complicated or rarely-used features of C++, for example, pointers, operator overloading, etc.
    - Java was Simple till Java 1.4. Later many new features are added in language to make it powerful (but complex too).
2. Object Oriented
    - Java is a object-oriented programming language.
    - Almost the "Everything is an Object" paradigm. All program code and data reside within objects and classes.
    - The object model in Java is simple and easy to extend.
    - Java comes with an extensive set of classes, arranged in packages that can be used in our programs through inheritance.
    - The basic concepts of OOPs are:
        - Object
        - Class
        - Abstraction
        - Encapsulation
        - Inheritance
        - Composition
        - Polymorphism
3. Distributed
    - Java is designed to create distributed applications connected over the network.
    - Java applications can access remote objects on the Internet as easily as they can do in the local system.
    - Java is designed for the distributed environment of the Internet. It handles TCP/IP protocols.
4. Compiled and Interpreted
    - Usually, a computer language is either compiled or Interpreted. Java combines both this approach and makes it a two-stage system.
    - Compiled: Java enables the creation of cross-platform programs by compiling them into an intermediate representation called Java Bytecode.
    - Interpreted: Bytecode is then interpreted, which generates machine code that can be directly executed by the machine/CPU.
5. Robust
    - It provides many features that make the program execute reliably in a variety of environments.
    - Java is a strictly typed language. It checks code both at compile time and runtime.

    - Java takes care of all memory management problems with garbage collection.
    - Exception handling captures all types of serious errors and eliminates any risk of crashing the system.
6. Secure
    - When a Java Compatible Web browser is used, downloading applets can be done safely without fear of infection or malicious intent.
    - Java achieves this protection by confining a Java program to the Java execution environment and not allowing it to access other parts of the computer.
7. Architecture Neutral
    - Java language and Java Virtual Machine helped in achieving the goal of WORA - Write (Compile) Once Run Anywhere.
    - Java byte code is interpreted and converted into CPU machine code/native code. So Java byte code can execute on any CPU architecture (on which JVM is available) like x86, SPARC, PPC, MIPS, etc.
8. Portable
    - Java is portable because of the Java Virtual Machine (JVM). The JVM is an abstract computing machine that provides a runtime environment for Java programs to execute.
    - The JVM provides a consistent environment for Java programs to run on, regardless of the underlying operating system. Java program can be written on one device and run on any other device with a JVM installed, without any changes or modifications.
9. High Performance
    - Java performance is high because of the use of bytecode.
    - The bytecode was used so that it can be efficiently translated into native machine code by JIT compiler (in JVM).
10. Multithreaded
    - Multithreaded Programs handled multiple tasks simultaneously (within a process), which was helpful in creating interactive, networked programs.
    - Java supports multi-process/thread communication and synchronization.
11. Dynamic
    - Java is capable of linking in new class libraries, methods, and objects.
    - Java classes has run-time type information (reflection) that is used to verify and resolve accesses to objects/members at runtime. This makes it possible to dynamically link code in a safe and expedient manner.

CLASSPATH

- CLASSPATH: Contains set of directories separated by ; (Windows) or : (Linux).
    - Java's environment variable by which one can inform Java compiler, application launcher, JVM and other Java tools about the directories in which Java classes/packages are kept.
    - CLASSPATH variable can be modified using "set" command (Windows) or "export" command (Linux).
        - Windows cmd> set CLASSPATH=\path\to\set;%CLASSPATH%
        - Linux terminal> export CLASSPATH=/path/to/set:$CLASSPATH
    - To display CLASSPATH variable
        - Windows cmd> set CLASSPATH
        - Linux terminal> echo $CLASSPATH
- Compilation and Execution (source code in "src" directory and .class file in "bin" directory)

    - terminal> cd \path\of\src directory
    - terminal> javac -d ..\bin Program.java
    - terminal> set CLASSPATH=..\bin
    - terminal> java Program

Console Input/Output

- Java has several ways to take input and print output. Most popular ways in Java 8 are given below:
- Using java.util.Scanner and System.out

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter name: ");
String name = sc.nextLine();
System.out.print("Enter age: ");
int age = sc.nextInt();
System.out.println("Name: " + name + ", Age: " + age);
System.out.printf("Name: %s, Age: %s\n", name, age);
```

Language Fundamentals

Naming conventions

- Names for variables, methods, and types should follow Java naming convention.
- Camel notation for variables, methods, and parameters.
    - First letter each word except first word should be capital.
    - For example:

```
public double calculateTotalSalary(double basicSalary, double
incentives) {
    double totalSalary = basicSalary + incentives;
    return totalSalary;
}
```

- Pascal notation for type names (i.e. class, interface, enum)
    - First letter each word should be capital.
    - For example:

```
class CompanyEmployeeManagement {
    // ...
}
```

- Package names must be in lower case only.
    - For example: javax.servlet.http;
- Constant fields must be in upper case only.
    - For example:

```
final double PI = 3.14;
final int WEEKDAYS = 7;
final String COMPANY_NAME = "Sunbeam Infotech";
```

Keywords

- Keywords are the words whose meaning is already known to Java compiler.
- These words are reserved i.e. cannot be used to declare variable, function or class.
- Java 8 Keywords
    1. abstract - Specifies that a class or method will be implemented later, in a subclass
    2. assert - Verifies the condition. Throws error if false.
    3. boolean- A data type that can hold true and false values only
    4. break - A control statement for breaking out of loops.
    5. byte - A data type that can hold 8-bit data values
    6. case - Used in switch statements to mark blocks of text
    7. catch - Catches exceptions generated by try statements
    8. char - A data type that can hold unsigned 16-bit Unicode characters
    9. class - Declares a new class
    10. continue - Sends control back outside a loop
    11. default - Specifies the default block of code in a switch statement
    12. do - Starts a do-while loop
    13. double - A data type that can hold 64-bit floating-point numbers
    14. else - Indicates alternative branches in an if statement
    15. enum - A Java keyword is used to declare an enumerated type. Enumerations extend the base class.
    16. extends - Indicates that a class is derived from another class or interface
    17. final - Indicates that a variable holds a constant value or that a method will not be overridden
    18. finally - Indicates a block of code in a try-catch structure that will always be executed
    19. float - A data type that holds a 32-bit floating-point number
    20. for - Used to start a for loop
    21. if - Tests a true/false expression and branches accordingly
    22. implements - Specifies that a class implements an interface
    23. import - References other classes
    24. instanceof - Indicates whether an object is an instance of a specific class or implements an interface
    25. int - A data type that can hold a 32-bit signed integer
    26. interface- Declares an interface
    27. long - A data type that holds a 64-bit integer
    28. native - Specifies that a method is implemented with native (platform-specific) code
    29. new - Creates new objects
    30. null - This indicates that a reference does not refer to anything
    31. package - Declares a Java package
    32. private - An access specifier indicating that a method or variable may be accessed only in the class it’s declared in

33. protected - An access specifier indicating that a method or variable may only be accessed in the class it's declared in (or a subclass of the class it's declared in or other classes in the same package)
34. public - An access specifier used for classes, interfaces, methods, and variables indicating that an item is accessible throughout the application (or where the class that defines it is accessible)
35. return - Sends control and possibly a return value back from a called method
36. short - A data type that can hold a 16-bit integer
37. static - Indicates that a variable or method is a class method (rather than being limited to one particular object)
38. strictfp - A Java keyword is used to restrict the precision and rounding of floating-point calculations to ensure portability.
39. super - Refers to a class's base class (used in a method or class constructor)
40. switch - A statement that executes code based on a test value
41. synchronized - Specifies critical sections or methods in multithreaded code
42. this - Refers to the current object in a method or constructor
43. throw - Creates an exception
44. throws - Indicates what exceptions may be thrown by a method
45. transient - Specifies that a variable is not part of an object's persistent state
46. try - Starts a block of code that will be tested for exceptions
47. void - Specifies that a method does not have a return value
48. volatile - This indicates that a variable may change asynchronously
49. while - Starts a while loop
50. goto, const - Unused keywords (Reserved words)
51. true, false, null - Literals (Reserved words)

Data types

- Data type describes:
  - Memory is required to store the data
  - Kind of data memory holds
  - Operations to perform on the data
- Java is strictly type checked language.
- In java, data types are classified as:
  - Primitive types or Value types
  - Non-primitive types or Reference types

```
Data types
    |- Primitive types (Value types)
    |      |- Boolean: boolean
    |      |- Character: char
    |      |- Integral: byte, short, int, long
    |      |- Floating-point: float, double
    |
    |- Non-Primitive types (Reference types)
           |- class
           |- interface
           |- enum
           |- Array
```

1. boolean (size is not specified)
2. byte (size is 1 byte)
3. char (size is 2 bytes)
4. short (size is 2 bytes)
5. int (size is 4 bytes)
6. float (size is 4 bytes)
7. double (size is 8 bytes)
8. long (size is 8 bytes)

- primitive types(boolean, byte, char, short, int ,float, double, long) are not classes in Java.

```
Stack<int> s1 =  new Stack<int>( ); //Not OK
Stack<Integer> s1 =  new Stack<Integer>( ); //OK
```

Datatype	Detail	Default	Memory needed (size)	Examples	Range of Values
boolean	It can have value true or false, used for condition and as a flag.	false	1 bit	true, false	true or false
byte	Set of 8 bits data	0	8 bits	NA	-128 to 127
char	Used to represent chars	\u0000	16 bits	“a”, “b”, “c”, “A” and etc.	Represents 0-256 ASCII chars
short	Short integer	0	16 bits	NA	-32768-32768
int	integer	0	32 bits	0, 1, 2, 3, -1, -2, -3	-2147483648 to 2147483647-
long	Long integer	0	64 bits	1L, 2L, 3L, -1L, -2L, -3L	-9223372036854775807 to 9223372036854775807
float	IEEE 754 floats	0.0	32 bits	1.23f, -1.23f	Upto 7 decimal
double	IEEE 754 floats	0.0	64 bits	1.23d, -1.23d	Upto 16 decimal

- Widening: We can convert state of object of narrower type into wider type. it is called as "widening".

```
int num1 = 10;
double num2 = num1; //widening
```

- Narrowing: We can convert state of object of wider type into narrower type. It is called "narrowing".

```
double num1 = 10.5;
int num2 = (int) num1; //narrowing
```

- Rules of conversion
  - source and destination must be compatible i.e. destination data type must be able to store larger/equal magnitude of values than that of source data type.
  - Rule 1: Arithmetic operation involving byte, short automatically promoted to int.
  - Rule 2: Arithmetic operation involving int and long promoted to long.
  - Rule 3: Arithmetic operation involving float and long promoted to float.
  - Rule 4: Arithmetic operation involving double and any other type promoted to double.
- Type Conversions

Literals

- Six types of Literals:
  - Integral Literals
  - Floating-point Literals
  - Char Literals
  - String Literals
  - Boolean Literals
  - null Literal

Integral Literals

- Decimal: It has a base of ten, and digits from 0 to 9.
- Octal: It has base eight and allows digits from 0 to 7. Has a prefix 0.
- Hexadecimal: It has base sixteen and allows digits from 0 to 9 and A to F. Has a prefix 0x.
- Binary: It has base 2 and allows digits 0 and 1.
- For example:

```
int x = 65; // decimal const don't need prefix
int y = 0101; // octal values start from 0
int z = 0x41; // hexadecimal values start from 0x
int w = 0b01000001; // binary values start with 0b
```

- Literals may have suffix like U, L.
  - L -- represents long value.

```
long x = 123L; // long const assigned to long variable
long y = 123; // int const assigned to long variable -- widening
```

Floating-Point Literals

- Expressed using decimal fractions or exponential (e) notation.

- Single precision (4 bytes) floating-point number. Suffix f or F.
- Double precision (8 bytes) floating-point number. Suffix d or D.
- For example:

```
float x = 123.456f;
float y = 1.23456e+2;   // 1.23456 x 10^2 = 123.456
double z = 3.142857d;
```

Char Literals

- Each char is internally represented as integer number - ASCII/Unicode value.
- Java follows Unicode char encoding scheme to support multiple langauges.
- For example:

```
char x = 'A';       // char representation
char y = '\101';    // octal value
char z = '\u0041';  // unicode value in hex
char w = 65;        // unicode value in dec as int
```

- There are few special char literals referred as escape sequences.
  - \n -- newline char -- takes cursor to next line
  - \r -- carriage return -- takes cursor to start of current line
  - \t -- tab (group of 8 spaces)
  - \b -- backspace -- takes cursor one position back (on same line)
  - ' -- single quote
  - " -- double quote
  - \ -- prints single \
  - \0 -- ascii/unicode value 0 -- null character

String Literals

- A sequence of zero or more unicode characters in double quotes.
- For example:

```
String s1 = "Sunbeam";
```

Boolean Literals

- Boolean literals allow only two values i.e. true and false. Not compatible with 1 and 0.
- For example:

```
boolean b = true;
boolean d = false;
```

Null Literal

- "null" represents nothing/no value.
- Used with reference/non-primitive types.

```
String s = null;
Object o = null;
```

Variables

- A variable is a container which holds a value. It represents a memory location.
- A variable is declared with data type and initialized with another variable or literal.
- In Java, variable can be
  - Local: Within a method -- Created on stack.
  - Non-static/Instance field: Within a class - Accessed using object.
  - Static field: Within a class - Accessed using class-name.

Operators

- Java divides the operators into the following catgories:
  - Arithmetic operators: +, -, *, /, %
  - Assignment operators: =, +=, -=, etc.
  - Comparison operators: ==, !=, <, >, <=, >=, instanceof
  - Logical operators: &&, ||, !
    - Combine the conditions (boolean - true/false)
  - Bitwise operators: &, |, ^, ~, <<, >>, >>>
  - Misc operators: ternary ?:, dot .
    - Dot operator: ClassName.member, objName.member.

- Operator precedence and associativity

Operator	Description	Associativity
++ --	unary postfix increment unary postfix decrement	right to left
++ -- + - ! ~ (*type*)	unary prefix increment unary prefix decrement unary plus unary minus unary logical negation unary bitwise complement unary cast	right to left
* / %	multiplication division remainder	left to right
+ -	addition or string concatenation subtraction	left to right
<< >>>>>>>	left shift signed right shift unsigned right shift	left to right
< <= > >= instanceof	less than less than or equal to greater than greater than or equal to type comparison	left to right
== !=	is equal to is not equal to	left to right
&	bitwise AND boolean logical AND	left to right

Wrapper types

- In Java primitive types are not classes. So their variables are not objects.
- Java has wrapper class corresponding to each primitive type. Their variables are objects.
- All wrapper classes are final classes i.e we cannot extend it.
- All wrapper classes are declared in java.lang package.

```
Object
    |- Boolean
    |- Character
    |- Number
        |- Byte
        |- Short
        |- Integer
        |- Long
```

```
            |- Float
            |- Double
```

- For every primitive, we get class in Java. It is called Wrapper class.

1. boolean => java.lang.Boolean
2. byte => java.lang.Byte
3. char => java.lang.Character
4. short => java.lang.Short
5. int => java.lang.Integer
6. float => java.lang.Float
7. double => java.lang.Double
8. long => java.lang.Long

- Applications of wrapper classes
  - Use primitive values like objects

```java
// int 123 converted to Integer object holding 123.
Integer i = new Integer(123);
```

  - Convert types

```java
Integer i = new Integer(123);
byte b = i.byteValue();
long l = i.longValue();
short s = i.shortValue();
double d = i.doubleValue();
String str = i.toString();

String val = "-12345";
int num = Integer.parseInt(val);
```

  - Get size and range of primitive types

```java
System.out.printf("int size: %d bytes = %d bits\n", Integer.BYTES,
Integer.SIZE);
System.out.printf("int max: %d, min: %d\n", Integer.MAX_VALUE,
Integer.MIN_VALUE);
```

  - Helper/utility methods

```java
System.out.println("Sum = " + Integer.sum(22, 7));
System.out.println("Max = " + Integer.max(22, 7));
System.out.println("Min = " + Integer.min(22, 7));
```

Boxing

- Converting from value (primitive) type to reference type.

```
int x = 123;
Integer y = new Integer(x); // boxing
```

- Java 5 allows auto-conversion from primitive type to corresponding wrapper type. This is called as "auto-boxing".

```
int x = 123;
Integer y = x; // auto-boxing
```

Unboxing

- Converting from reference type to value (primitive) type.

```
Integer y = new Integer(123);
int x = y.intValue(); // unboxing
```

- Java 5 allows auto-conversion from wrapper type to corresponding value type. This is called as "auto-unboxing".

```
Integer y = new Integer(123);
int x = y; // auto-unboxing
```

Control Statements

- By default, Java statements are executed sequentially i.e. statements are executed in order in which they appear in code.
- Java also provide statements to control the flow of execution. These are called as control statements.
- Types of control flow statements.
  - Decision Making statements
    - if statements
    - switch statement
  - Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop

        - labeled loop
    - Jump statements
        - break statement
        - continue statement
        - return statement
- Being structured programming language, control flow statements (blocks) can be nested within each other.

if statements

- In Java, conditions are boolean expressions that evaluate to true or false.
- Program execution proceeds depending on condition (true or false).
- Syntax:

```
if(condition) {
    // execute when condition is true
}
```

```
if(condition) {
    // execute when condition is true
}
else {
    // execute when condition is false
}
```

```
if(condition1) {
    // execute when condition1 is true
}
else if(condition2) {
    // execute when condition2 is true
}
else {
    // execute when condition no condition is true
}
```

switch statements

- Selects a code block to be executed based on given expression.
- The expression can be integer, character or String type.

```
switch (expression) {
    case value1:
        // executed when expression equals value1
    break;
```

```
    case value2:
        // executed when expression equals value2
    break;
    // ...
    default:
        // executed when expression not equals any case constant
}
```

- We can use String constants/expressions for switch case in Java (along with integer and char types).

```
String course = "DAC";
switch(course) {
case "DAC":
    System.out.println("Welcome to DAC!");
    break;
case "DMC":
    System.out.println("Welcome to DMC!");
    break;
case "DESD":
    System.out.println("Welcome to DESD!");
    break;
default:
    System.out.println("Welcome to C-DAC!");
}
```

do-while loop

- Executes loop body and then evaluate the condition.
- If condition is true, loop is repeated again.

```
do {
    // body
} while(condition);
```

- Typically used to implement menu-driven program with swicth-case.

while loop

- Evaluate the condition and repeat the body if condition is true.

```
initialization;
while(condition) {
    body;
    modification;
}
```

for loop

- Initialize, evaluate the condition, execute the body if condition is true and then execute modification statement.

```
for(initialization; condition; modification) {
    body;
}
```

for-each loop

- Execute once for each element in array/collection.

```
int[] arr = { 11, 22, 33, 44 };
for(int i: arr)
    System.out.println(i);
```

break statement

- Stops switch/loop execution and jump to next statement after the switch/loop.

```
initialization;
while(condition) {
    body;
    if(stop-condition)
        break;
    modification;
}
statements after loop;
```

continue statement

- Jumps to next iteration of the loop.

```
initialization;
while(condition) {
    if(skip-condition)
        continue;
    body;
    modification;
}
statements after loop;
```

Labeled loops

- In case of nested loop, break and continue statements affect the loop in which they are placed.
- Labeled loops overcome this limitation. Programmer can choose the loop to be affected by break/continue statement.
- Labels can be used with while/for loop.

```
outer: for(int i=1; i<=3; i++) {
    middle: for(int j=1; j<=3; j++) {
        inner: for(int k=1; k<=3; k++) {
            if(i==j && j==k && i==K)
                break middle;
            System.out.printf("%d, %d, %d\n", i, j, k);
        }
    }
}
```

```
2 1 1
2 1 2
2 1 3
2 2 1
3 1 1
3 1 2
3 1 3
3 2 1
3 2 2
3 2 3
3 3 1
3 3 2
```

Ternary operator

- Ternary operator/Conditional operator

```
condition? expression1 : expression2;
```

- Equivalent if-else code

```
if(condition)
    expression1;
else
    expression2;
```

- If condition is true, expression1 is executed and if condition is false, expression2 is executed.

```
a = 10;
b = 7;
max = (a > b) ? a : b;
```

```
a = 10;
b = 17;
max = (a > b) ? a : b;
```

Java methods

- A method is a block of code (definition). Executes when it is called (method call).
- Method may take inputs known as parameters.
- Method may yield a output known as return value.
- Method is a logical set of instructions and can be called multiple times (reusability).

```
class ClassName {
    public static ret-type staticMethod(parameters) {
        // method body
    }

    public ret-type nonStaticMethod(parameters) {
        // method body
    }

    public static void main(String[] args) {
        // ...
        res1 = ClassName.staticMethod(arguments);
        ClassName obj = new ClassName();
        res2 = obj.nonStaticMethod(arguments);
    }
}
```

Class and Object

- Class is collection of logically related data members ("fields"/attributes/properties) and the member functions ("methods"/operations/messages) to operate on that data.
- A class is user defined data type. It is used to create one or more instances called as "Objects".
- Class is blueprint/prototype/template of the object; while Object is an instance of the class.
- Class is logical entity and Object represent physical real-world entity.
- Class represents group of such instances which is having common structure and common behavior.

- class is an imaginary entity.
  - Example: Car, Book, Laptop etc.
  - Class implementation represents encapsulation.
  - e.g. Human is a class and You are one of the object of the class.

```java
class Human {
    int age;
    double weight;
    double height;
    // ...
    void walk() { ... }
    void talk() { ... }
    void think() { ... }
    // ...
}
```

- Since class is non-primitive/reference type in Java, its objects are always created on heap (using new operator). Object creation is also referred as "Instantiation" of the class.

```java
Human obj = new Human();
obj.walk();
```

Instance

```
- Definition
    1. In java, object is also called as instance
    2. An entity, which is having physical existance is called as instance.
    3. An entity, which is having state, behavior and identity is called as
instance.
- instance is a real time entity.
- Example: "Tata Nano", "Java Complete Reference", "MacBook Air" etc.
```

- Types of methods in a Java class.
  - Methods are at class-level, not at object-level. In other words, same copy of class methods is used by all objects of the class.
  - Parameterless Constructor
    - In Java, fields have default values if unintialized. Primitive types default value is usually zero (refer data types table) and Reference type default value is null. Constructor should initialize fields to the desired values.
    - Has same name as of class name and no return type. Automatically called when object is created (with no arguments).
    - If no construcor defined in class, Java compiler provides a default construcor (Parameterless).

    ```java
    Human obj = new Human();
    ```

  - Parameterized Constructor

        - Constructor should initialize fields to the given values.
        - Has same name as of class name and no return type. Automatically called when object is created (with arguments).

```
Human obj = new Human(40, 76.5, 5.5);
```

    - Inspectors/Getters
        - Used to get value of the field outside the class.

```
double ht = obj.getHeight();
```

    - Mutators/Setters
        - Used to set value of the field from outside the class.
        - It modifies state of the object.

```
obj.setHeight(5.5);
```

        - Getter/setters provide "controlled access" to the fields from outside the class.
    - Facilitators
        - Provides additional functionalities like Console IO, File IO.

```
obj.display();
```

    - Other methods/Business logic methods

```
obj.talk();
```

- Executing a method on object is also interpreted as
    - Calling member function/method on object.
    - Invoking member function/method on object.
    - Doing operation on the object.
    - Passing message to the object.
- Each object has
    - State: Represents values of fields in the object.
    - Behaviour: Represents methods in the class and also depends on Object state.
    - Identity: Represents uniqueness of the object.
- Object created on heap using new operator is anonymous.

```
new Human().talk();
```

- Assigning reference doesn't create new object.

```
Human h1 = new Human(...);
Human h2 = h1;
```

How to get system date in Java?

- Using Calender class:

```
Calendar c = Calendar.getInstance();
//int day = c.get( Calendar.DAY_OF_MONTH );
int day = c.get( Calendar.DATE );
int month = c.get( Calendar.MONTH) + 1;
int year = c.get( Calendar.YEAR );
```

Initialization

```
int num1 = 10;      //Initialization
int num2 = num1;    //Initialization
```

- Initialization is the process of storing value inside variable during its declaration.
- We can initialize any variable only once.

Assignment

```
int num1 = 10;  //Initialization
//int num1 = 20;  //Not OK
num1 = 20;  //OK:   Assignment
num1 = 30;  //OK:   Assignment
```

- Assignment is the process of storing value inside variable after its declaration.
- We can do assignment multiple times.

Constructor

- It is a method of class which is used to initialize instance.
- Constructor is a special syntax of Java because:
    1. Its name and class name is always same.
    2. It doesn't have any return type
    3. It is designed to call implicitly.
    4. In the lifetime on instance, it gets called only once.

```
public Date(  ){    //Constructor of the class
    System.out.println("Inside constructor");
    Calendar c = Calendar.getInstance( );
    this.day = c.get( Calendar.DATE );
    this.month = c.get( Calendar.MONTH ) + 1;
    this.year = c.get( Calendar.YEAR );
}
```

- We can not call constructor on instance explicitly.

```
Date date = new Date(); //OK
date.Date( );   //Not OK
```

- We can use any access modifer on constructor.
- If constructor is public then we can create instance of a class inside method of same class as well as different class.
- If constructor is private then we can create instance of class inside method of same class only.
- Types of constructor in Java:
    1. Parameterless constructor.
    2. Parameterized constructor.
    3. Default constructor.

Parameterless constructor

- A constructor which do not have any parameter is called parameterless constructor.

```
    public Date(  ){   //Parameterless constructor
        Calendar c = Calendar.getInstance( );
        this.day = c.get( Calendar.DATE );
        this.month = c.get( Calendar.MONTH ) + 1;
        this.year = c.get( Calendar.YEAR );
    }
```

- If we create instance W/O passing arguments then parameterless constructor gets called.

```
Date dt1 = new Date( );  //OK
```

- In the above code, parameterless constructor will call.

Parameterized constructor

- Constructor of a class which take parameters is called parameterized constructor.

```
public Date( int day, int month, int year ){    //Parameterized constructor
    this.day = day;
    this.month = month;
    this.year = year;
}
```

- If we create instance by passing arguments then parameterized constructor gets called.

```
Date date = new Date( 23, 7, 1983 );    //OK
```

- Constructor calling sequence depends on order of instance creation.

Default constructor

- If we do not define any constructor(no parameterless, no parameterized) inside class then compiler generate one constructor for the class by default. It is called default constructor.
- Compiler generated default constructor is zero parameter i.e parameterless constructor.
- Compiler never generate default parameterized constructor. It means that, if we want to create instance with arguments then we must define parameterized constructor inside class.

Constructor chaining

- In Java, we can call constructor from another constructor. It is called constructor chaining.
- For constructor chaining, we should use this statement.
- this statement must be first statement inside constructor body.

```
class Date{
    private int day;    //Default value is 0
    private int month;  //Default value is 0
    private int year;   //Default value is 0

    public Date(  ){    //Parameterless Constructor
        this( 12, 8, 2022);    //Constructor chaining
    }
    public Date( int day, int month, int year ){    //Parameterized constructor
        this.day = day;
        this.month = month;
        this.year = year;
    }
}
```

- Using constructor chaning, we can reduce developers efforts.

this reference

- If we call non static method on instance then non static method get this reference.

- this reference contains reference of current/calling instance.
- Using this reference, non static method and non static field can communicate with each other. Hence this reference is considered as a link/connection between them.
- this reference is considered as a first parameter to the method. Hence it gets space once per method call on Java Stacks.
- We can not use this reference to access local variable. We should use this reference to access non static field/method.
- Use of this reference to access non static field/method is optional.
- If name of the local variable and name of field is same then to refer field we should use this reference.

```
class Complex{
    private int real;
    public void setReal( int real ){
        this.real = real;
    }
}
```

- Definition:
    - this reference is implicit parameter available inside every non static method of the class which is used to store reference of current/calling instance.

OOPS concepts

- Following members do not get space inside instance
    1. Method parameter(e.g this reference, args etc)
    2. Method local variable
    3. Static field
    4. Static as well as non static method
    5. Constructor
- Only non static field get space inside instance.
- If we declare non static field inside class then it gets space once per instance according their order of declaration.

```
class Test{
    private int num1;
    private int num2;
    private int num3;
}
class Program{
    public static void main(String[] args) {
        Test t1 = new Test( );
        Test t2 = new Test( );
        Test t3 = new Test( );
    }
}
```

- Method do not get space inside instance. All the instances of same class share single copy of method declared inside class.
- Instances can share method by passing reference of the instance as argument to the method.

Object Oriented Programming Structure / System(OOPS)

- OOPS is not a syntax. Rather it is a process or a programming methodology that we can use to solve real world use cases / problems. In other words, oops is an object oriented thought process.
- Alan Kay is inventor of oops.
- Ref: https://medium.com/javascript-scene/the-forgotten-history-of-oop-88d71b9b2d9f
- Object Oriented Analysis and Design with application: Grady Booch
- According to Grady Booch, there are 4 major and 3 minor pillars of oops.

4 Major pillars / parts / elemets

1. Abstraction : To achive simplicity
2. Encapsulation : To achive data hiding and data security
3. Modularity : To minimize module dependency
4. Hierarchy : To achive reusability

- By major, we mean that a language without any one of these elements is not object oriented.

3 Minor pillars / parts / elemets

1. Typing : To minimize maintenance of the system.
2. Concurrency : To utilize hardware resources efficiently.
3. Persistence : To maintain state of the instance on secondary storage

- By minor, we mean that each of these elements is a useful, but not essential.

Abstraction

- It is a major pillar of oops.
- Abstraction is the process of getting essential things from system/object.
- Abstraction focuses on the essential characteristics of som object, relative to the perspective of viewer. In simple word, abstraction changes from user to user.
- Abstraction describes external behavior of an instance.
- Creating instance and calling methods on it represents abstraction programatically.

```
Scanner sc = new Scanner( System.in );
String name = sc.nextLine( );
int empid = sc.nextInt( );
float salary = sc.nextFloat( );
```

```
Complex c1 = new Complex();
c1.acceptRecord( );
c1.printRecord( );
```

- Using abstraction, we can achive simplicity.

Encapsulation

- It is a major pillar of oops.
- Definition:
    1. To achive abstraction, we need to provide some implementation. This implementation of abstraction is called encapsulation.
    2. Binding of data and code together is called as encapsulation.
- Hiding represents encapsulation.

```
class Complex{
    /* Data*/
    private int real;
    private int imag;
    public Complex( ){
        this.real = 0;
        this.imag = 0;
    }
    /*Code*/
    public void acceptRecord( ){
        Scanner sc = new Scanner( System.in)
        //TODO: accept record for real and imag
    }
    public void printRecord( ){
        //TODO: print state of real and imag
    }
}
class Program{
    public static void main(String[] args) {
        //Abstraction
        Complex c1 = new Complex();
        c1.acceptRecord( );
        c1.printRecord( );
    }
}
```

- Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.
- Class implementation represents encapsulation.
- Use of encapsulation:
    1. To achive abstraction
    2. To achive data hiding(also called as data encapsulation).
- Process of declaring field private is called as data hiding.
- Data hiding help to achive data security.

Modularity

- It is a major pillar of oops
- Modularity is the process of designing and developing complex system using small parts/modules.
- Main purpose of modularity is to minimize module dependancy.
- Using .jar file, we can achive modularity in Java.

Hierarchy

- It is a major pillar of oops.
- Level/order/ranking of abstraction is called as hierarchy.
- Using hierarchy, we can achive code reusability.
- Application of reusability:
    1. To reduce development time.
    2. To reduce development cost
    3. To reduce develpers effort.
- Types of hierarchy:
    1. Has-a => Association
    2. Is-a => Generalization(is also called as inheritance)
    3. Use-a => Dependancy
    4. Create-a => Instantiation

Typing

- It is a minor pillar of oops.
- It is also called as polymorphism(poly(many) + morphism(forms/behavior)).
- polymorphism is a Greek word.
- An ability of any instance to take multiple forms is called as polymorphism.
- Using typing/polymorphism, we can reduce maintenance of the application.
- In Java, we can achive polymorphism using two ways:
    1. Method overloading (It represents compile time polymorphism)
    2. Method overriding (It represents run time polymorphism)
- In Java, runtime polymorphism is also called as dynamic method dispatch

Concurrency

- It is a minor pillar of oops.
- Concurrency is the process of executing multiple task simultaneously.
- Using thread, we can achive Concurrency in Java.
- Using Concurrency, we can utilize H/W resources efficiently.

Persistance

- It is a minor pillar of oops.
- Process of maintaining state of instance inside file / database is called as Persistance.
- We can implement it using file handing(serialization / derserialization) and database programming(JDBC).

Class

```
Date dt2 = dt1;  //Shallow copy of references

Date dt3 = new Date( );
```

```
Syntax of creating instance/object:
//ClassName refName = new ClassName( );

1. Date dt1 = new Date( );

2. Employee emp = new Employee( );

3. Account acc; //reference
   acc = new Account( );   //Instance
```

```
Date birthDate = new Date( );
```

```Java
// for-each loop is used to access elements one by one from any collection
// print each element in array
int arr[] = { 11, 22, 33, 44, 55 };

for(int ele : arr) {
    System.out.println(ele);
}
```

11 22 33 44 55

getters/setters provide controlled access of private fields outside the class.

controlled access

- write setters only for modifiable fields (logically)
- write getters only for fields to be read
- setters can have checks for valid values

java.lang.Object
java.lang.Boolean
<<final>>
java.lang.Number
<<Abstract>>
java.lang.Character
<<final>>
java.lang.Byte
<<final>>
java.lang.Short
<<final>>
java.lang.Integer
<<final>>
java.lang.Float
<<final>>
java.lang.Double
<<final>>
java.lang.Long
<<final>>

```
Employee emp = new Employee( );
```

Employee emp = new Employee();

An object has state, exhibits some well-defined behavior, and has a unique identity.

Driver
Passenger
Bus
Mechanic

Abstraction
Abstraction
PURR PURR PURR
Puss
Kidney
stomach
intestine
heart
lung
bladder
liver
Cat
Lady
Doctor
VETERINARY SURGEON
Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

Encapsulation hides the details of the implementation of an object.

Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior. Encapsulation is most often achieved through information hiding (not just data hiding), which is the process of hiding all the secrets of an object that do not contribute to its essential characteristics; typically, the structure of an object is hidden, as well as the implementation of its methods. "No part of a complex system should depend on the internal details of any other part" [50]. Whereas abstraction "helps people to think about what they are doing," encapsulation "allows program changes to be reliably made with limited effort" [51].

day02 - demo04/src/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
public class Program02 {
    public static void main(String[] args) {
        int a = 123;
        // convert primitive int to wrapper Integer (boxing)
        Integer b = new Integer(a);

        // convert wrapper Integer to primitive int (unboxing)
        int d = b.intValue();

        System.out.println("a = " + a + ", b = " + b + ", d = " + d);
    }
}
```

Problems Javadoc Declaration Console

<terminated> Program02 (2) [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Jan 30, 2024, 12:53:32 PM – 12:53:32 PM) [pid: 3176]

```
a = 123, b = 123, d = 123
```

Writable | Smart Insert | 11 : 68 : 306

Search 12:53 PM

day02 - demo04/src/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
public class Program02 {
    public static void main(String[] args) {
        int a = 123;
        // convert primitive int to wrapper Integer -- boxing
        Integer b = new Integer(a);
        // convert primitive int to wrapper Integer -- auto-boxing
        Integer c = a;

        // convert wrapper Integer to primitive int
        int d = b.intValue();
        // convert wrapper Integer to primitive int -- auto-unboxing
        int e = c;

        System.out.println("a = " + a + ", b = " + b + ", c = " + c + ", d = " + d + ", e = " + e);
    }
}
```

Problems Javadoc Declaration Console

<terminated> Program02 (2) [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Jan 30, 2024, 12:58:53 PM – 12:58:53 PM) [pid: 4172]

```
a = 123, b = 123, c = 123, d = 123, e = 123
```

Writable | Smart Insert | 15 : 100 : 506

Search 12:59 PM

day03.md > # Core Java > ## Control Statements > ### Ternary operator

* Ternary operator/Conditional operator

```Java
condition? expression1 : expression2;
```

* Equivalent if-else code

```Java
if(condition)
    expression1;
else
    expression2;
```

* If condition is true, expression1 is executed and if condition is false, expression2 is executed.

```Java
a = 10;
b = 7;
max = (a > b) ? a : b;
```

true

```Java
a = 10;
b = 17;
max = (a > b) ? a : b;
```

false

Class and Object

* Class is collection of logically related data members ("fields"/attributes/properties) and the member functions ("methods"/operations/messages) to operate on that data.
* A class is user defined data type. It is used to create one or more instances called as "Objects".
* Class is blueprint/prototype/template of the object; while Object is an instance of the class.
* Class is logical entity and Object represent physical real-world entity.
    * e.g. Human is a class and You are one of the object of the class.

```Java
class Human {
    int age;
    double weight;
    double height;
    // ...
    void walk() { ... }
    void talk() { ... }
    void think() { ... }
    // ...
}
```

data members / fields / attributes

member functions / methods / operations

Human h1 = new Human(); -- object creation

* Since class is non-primitive/reference type in Java, its objects are always created on heap (using new operator). Object creation is also referred as "Instantiation" of the class.

```Java
```

public void acceptRecord( )
public void printRecord( )
Method Area
[Employee Instance]
[Employee Instance]
[Employee Instance]
name
"Sandeep"
String
empid
123
int
salary
25000.50
float
name
"Akash"
String
empid
456
int
salary
27000.50
float
name
"Deepak"
String
empid
789
int
salary
29500.50
float
[Heap]
emp1
Employee
emp2
Employee
emp3
Employee
[Java Stack]
[Reference of class Employee]
[Reference of class Employee]
[Reference of class Employee]

Java - Access modifiers

Visibility of class A members

- private members
- default members
- protected members
- public members

Access modifiers

public → everywhere

protected → in the package + inherited classes in other pkgs

default → only in the package

private → only in the class

Core Java

"this" reference

- "this" is implicit reference variable that is available in every non-static method of class which is used to store reference of current/calling instance.
- Whenever any non-static method is called on any object, that object is internally passed to the method and internally collected in implicit "this" parameter.
- "this" is constant within method i.e. it cannot be assigned to another object or null within the method.
- Using "this" inside method (to access members) is optional. However, it is good practice for readability.
- In a few cases using "this" is necessary.
    - Unhide non-static fields from local fields.

      ```
      double height; // "height" field
      void setHeight(double height) { // "height" parameter
          height = height; // ???
      }
      ```

    - Constructor chaining
    - Access instance of outer object.

null reference

- In Java, local variables must be initialized before use; otherwise it raise compiler error.

  ```
  int a;
  a++; // compiler error
  Human obj;
  obj.walk(); // compiler error
  ```

- In Java, reference can be initialized to null (if any specific object is not yet available). However reference must be initialized to appropriate object before its use; otherwise it will raise NullPointerException at runtime.

  ```
  Human h = null; // reference initialized to null
  h = new Human(); // reference initialized to the object
  h.walk(); // invoke the method on the object
  ```

  ```
  Human h = null;
  h.walk(); // NullPointerException
  ```

Constructor chaining

- Constructor chaining is executing a constructor of the class from another constructor (of the same class).

```
Human(int age, double height, double weight) {
    this.age = age;
    this.height = height;
    this.weight = weight;
}
Human() {
    this(0, 1.6, 3.3);
    // ...
}
```

- Constructor chaining (if done) must be on the very first line of the constructor.

OOP concepts

Abstraction

- Abstraction is getting essential details of the system.
- Abstraction change as per perspective of the user.
- Abstraction represents outer view of the system.
- Abstraction is based on interface/public methods of the class.

Encapsulation

- Combining information/state and complex logic/operations so that it will be easier to use is called as Encapsulation.
- Fields and methods are bound in a class so that user can create object and invoke methods (without looking into complex implementations).
- Encapsulation represents inner view of the system.
- Encapsulation and abstraction are complementary to each other.

Information hiding

- Members of class not intended to be visible outside the class can be restricted using access modifiers/specifiers.
- The "private" members can be accessed only within the class; while "public" members are accessible in as well as outside the class.
- Usually fields (data) are "private" and methods (operations) are "public".

Packages

- Packages makes Java code modular. It does better organization of the code.

- Package is a container that is used to group logically related classes, interfaces, enums, and other packages.

- Package helps developer:
  - To group functionally related types together.
  - To avoid naming clashing/collision/conflict/ambiguity in source code.
  - To control the access to types.
  - To make easier to lookup classes in Java source code/docs.
- Java has many built-in packages.
  - java.lang * --> Integer, System, String, ...
  - java.util --> Scanner, ArrayList, Date, ...
  - java.io --> FileInputStream, FileOutputStream, ...
  - java.sql --> Connection, Statement, ResultSet, ...
  - java.util.function --> Predicate, Consumer, ...
  - javax.servlet.http --> HttpServlet, ...
- Package Syntax
  - To define a type inside package, it is mandatory write package declaration statement inside .java file.
  - Package declaration statement must be first statement in .java file.
  - Types inside package called as packaged types; while others (in default package) are unpackaged types.
  - Any type can be member of single package only.
- It is standard practice to have multi-level packages (instead of single level). Typically package name is module name, dept/project name, website name in reverse order.

```java
package com.sunbeaminfo.modular.corejava;
```

```java
package com.sunbeaminfo.dac;
```

- Packages can be used to avoid name clashing. In other words, two packages can have classes/types with same name.

```java
package com.sunbeam.tree;

class Node {
    // ...
}

public class Tree {
    // ...
}
```

```java
package com.sunbeam.list;

class Node {
    // ...
}

public class List {
    // ...
}
```

Access modifiers (for types)

- default: When no specifier is mentioned.
  - The types are accessible in current package only.
- public: When "public" specifier is mentioned.
  - The types are accessible in current package as well as outside the package (using import).

Access modifiers (for type members)

- private: When "private" specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
- default: When no specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
  - The members are accessible in all classes in same package (obj.member OR ClassName.member).
- protected: When "protected" specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
  - The members are accessible in all classes in same package including its sub-classes (super.member, obj.member OR ClassName.member).
  - The members are accessible in sub classes outside the package including its sub-classes (super.member).
- public: When "public" specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
  - The members are accessible in all other classes (super.member, obj.member OR ClassName.member).
- Scopes
  - private (lowest)
  - default
  - protected
  - public (highest)

	default	private	protected	public
same class	yes	yes	yes	yes
same package subclass	yes	no	yes	yes
same package non-subclass	yes	no	yes	yes
different package subclass	no	no	yes	yes
different package non-subclass	no	no	no	yes

-

Array

- Array is collection of similar data elements. Each element is accessible using indexes (0 to n-1).
- In Java, array is non-primitive/reference type i.e. its object is created using new operator (on heap).
- Array size is fixed - given while creating array object. It cannot be modified later.
- Java array object holds its length and data elements.
- The array of primitive type holds values (0 if uninitialized) and array of non-primitive type holds references (null if uninitialized).
- Array of primitive types

```
int[] arr1 = new int[5];

int[] arr2 = new int[5] { 11, 22, 33 };
// error

int[] arr3 = new int[] { 11, 22, 33, 44, 55 };

int[] arr4 = { 11, 22, 33, 44 };
```

- Array of non-primitive types

```
Human[] arr5 = new Human[5];

Human[] arr6 = new Human[] {h1, h2, h3};
// h1, h2, h3 are Human references

Human[] arr7 = new Human[] {
    new Human(...),
    new Human(...),
    new Human(...)
};
```

- In Java, checking array bounds is responsibility of JVM. When invalid index is accessed, ArrayIndexOutOfBoundsException is thrown.
- Array types are
  - 1-D array
  - 2-D/Multi-dimensional array
  - Ragged array

1-D array

```
int[] arr = new int[5];

int[] arr = new int[4] { 10, 20, 30, 40 };
// error

int[] arr = new int[] { 11, 22, 33, 44, 55 };

int[] arr = { 11, 22, 33, 44 };

Human[] arr = new Human[5];

Human[] arr = new Human[] {h1, h2, h3};
```

- Individual element is accesses as `arr[i]`.

[Multi Dimensional Array]

File Edit Selection View Go Run Terminal Help private

day03.md | day04.md .\ | classwork.md U | day04.md C:\...\private

day04.md > # Core Java > ## Day 04 Agenda

```
# Core Java

## Day 04 Agenda
* Class and Object
    * this pointer
    * null reference
    * constructor chaining
* OOP concepts
    * Abstraction
    * Encapsulation
    * Information hiding
* Packages
* Access modifiers
* Arrays
    * 1-D array
    * 2-D array

## Class and Object

### "this" reference
* "this" is implicit reference variable that is available in every non-static method of class which is used
to store reference of current/calling instance
```

packages = Modularity

```
package project;

package list;
class Node {
    ...
}
public
class List {
    ...
}

package tree;
class Node {
    ...
}
public
class Tree {
    ...
}

package main;
import project.list.*;
import project.tree.*;
class Main {
    main() {
        ... Tree t = ...;
        List l = ...;
    }
}
```

package -- container for types (like class, interface, enum) and sub-packages.

packages -- avoids name clashing/confict.

when compiled, a directory is created for each package.

main* 0 0 0 Ln 13, Col 1 (12 selected) Spaces: 4 UTF-8 CRLF Markdown

Search 10:40 AM

demo02

This PC > Local Disk (C:) > Nilesh > onlinecourses > CJ-H-02 > day04 > demo02

Name	Date modified	Type	Size
main	2/1/2024 11:05 AM	File folder	
math	2/1/2024 11:04 AM	File folder	
Arithmetic.java	2/1/2024 11:00 AM	Java Source File	1 KB
Program.java	2/1/2024 11:02 AM	Java Source File	1 KB

when compiled, a directory is created for each package, and .class file created in it.

Arithmetic.java - Notepad

```java
package math;

public class Arithmetic {
    public void add(int a, int b) {
        int r = a + b;
        System.out.println(r);
    }

    public void subtract(int a, int b) {
        int r = a - b;
        System.out.println(r);
    }
}
```

Ln 11, Col 14 | 70% | Windows (CRLF) | UTF-8

Program.java - Notepad

```java
package main;

import math.Arithmetic;

class Program {
    public static void main(String[] args) {
        Arithmetic obj = new Arithmetic();
        obj.add(22, 7);
        obj.subtract(22, 7);
    }
}
```

Ln 3, Col 7 | 80% | Windows (CRLF) | UTF-8

Conventionally pkg name written in small case.

Windows PowerShell

```
PS C:\Nilesh\onlinecourses\CJ-H-02\day04\demo02> javac -d . Arithmetic.java

PS C:\Nilesh\onlinecourses\CJ-H-02\day04\demo02> javac -d . Program.java
PS C:\Nilesh\onlinecourses\CJ-H-02\day04\demo02>

PS C:\Nilesh\onlinecourses\CJ-H-02\day04\demo02> java Program
Error: Could not find or load main class Program
Caused by: java.lang.ClassNotFoundException: Program
PS C:\Nilesh\onlinecourses\CJ-H-02\day04\demo02> java main.Program
29
15
PS C:\Nilesh\onlinecourses\CJ-H-02\day04\demo02>
```

when class is in some package, always access with its full name i.e. packagename.ClassName.

4 items | 1 item selected

Search | 11:07 AM

geometry

File Home Share View

Pin to Quick access | Copy | Paste | Cut | Copy path | Paste shortcut | Move to | Copy to | Delete | Rename | New folder | New item | Easy access | Properties | Open | Edit | History | Select all | Select none | Invert selection

Clipboard | Organize | New | Open | Select

This PC > Local Disk (C:) > Nilesh > onlinecourses > CJ-H-02 > day04 > demo03 > sunbeam > geometry

Search geometry

Name	Date modified	Type	Size
dim2	2/1/2024 11:48 AM	File folder	
dim3	2/1/2024 11:47 AM	File folder	
Program.class	2/1/2024 11:48 AM	CLASS File	2 KB

Windows PowerShell | Windows PowerShell | Command Prompt

```
C:\Nilesh\onlinecourses\CJ-H-02\day04\demo03>javac -d . Box.java

C:\Nilesh\onlinecourses\CJ-H-02\day04\demo03>javac -d . Cylinder.java

C:\Nilesh\onlinecourses\CJ-H-02\day04\demo03>javac -d . Circle.java

C:\Nilesh\onlinecourses\CJ-H-02\day04\demo03>javac -d . Program.java

C:\Nilesh\onlinecourses\CJ-H-02\day04\demo03>java sunbeam.geometry.Program
Circle area: 153.958
Cylinder volume: 219.94
Box volume: 105.0

C:\Nilesh\onlinecourses\CJ-H-02\day04\demo03>
```

Select a file to preview.

3 items

Search 11:49 AM

day04 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java

```java
package com.sunbeam;

public class Program05 {
    public static void main(String[] args) {
        int[] arr1 = new int[5];

        int[] arr2 = new int[] { 11, 22, 33, 44 };

        int[] arr = { 10, 20, 30, 40, 50 };

        for(int i=0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

Problems Javadoc Declaration Console

<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 1, 2024, 12:58:05 PM – 12:58:06 PM) [pid: 16164]

```
10
20
30
40
50
```

Checking array bounds is responsibility of JVM.
If invalid index accessed, ArrayIndexOutOfBoundsException.

arr[i]
- access ith element from array
- 0 to length-1

for(int num : arr)
sysout(num);
array eles can be accessed using for each loop.

Writable | Smart Insert | 11 : 27 : 238

Search 12:58 PM

day04 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java

```java
    public static void main(String[] args) {
        int[] arr1 = new int[5];

        int[] arr2 = new int[] { 11, 22, 33, 44 };

        int[] arr = { 10, 20, 30, 40, 50 };

        for(int i=0; i < arr.length; i++)
            System.out.println(arr[i]);

        Scanner sc = new Scanner(System.in);

        double[] array = new double[3];
        System.out.println("Enter array elements: ");
        for(int i=0; i<array.length; i++)
            array[i] = sc.nextDouble();

        double sum = Program05.arraySum(array);

        System.out.println("Sum : " + sum);
    }
    public static double arraySum(double[] arr) {
        double total = 0.0;
        for(int i=0; i<arr.length; i++)
            total = total + arr[i];
        return total;
    }
```

Problems Javadoc Declaration Console

<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\o

```
10
20
30
40
50
Enter array elements:
1.1
3.3
5.5
Sum : 9.9
```

Search 1:11 PM

day05 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java × Human.java

```java
package com.sunbeam;

public class Program01 {
    public static void main(String[] args) {
        Human[] arr = new Human[4];

        arr[0] = new Human(40, 178, 80);
        arr[1] = new Human(12, 150, 40);
        arr[2] = new Human(12, 148, 30);
        arr[3] = new Human(35, 160, 60);

        for(int i=0; i<arr.length; i++)
            arr[i].display();
    }
}
```

Problems Javadoc Declaration Console ×

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 2, 2024, 9:38:12 AM – 9:38:13 AM) [pid: 5156]

```
Age: 40, Height: 178, Weight: 80
Age: 12, Height: 150, Weight: 40
Age: 12, Height: 148, Weight: 30
Age: 35, Height: 160, Weight: 60
```

When we create array of objects -- actually it creates array of references. By default, these references are null.

Programmer must initialize each reference (array element) before using it. Otherwise it will raise, NullPoinerException.,

Writable | Smart Insert | 10 : 41 : 270

Search 9:38 AM

day05 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java | Human.java | Program02.java

```java
ublic class Program02 {
    public static void main(String[] args) {
        int[][] arr = new int[3][4]; // by default all ints initialized to zero
        //int arr[][] = new int[3][4];
        //int[] arr[] = new int[3][4];

        //double[][] arr2 = new double[][] { {1.1, 2.2, 3.3}, {4.4, 5.5, 6.6} }; // 2 rows x 3 cols
        double[][] arr2 = { {1.1, 2.2, 3.3}, {4.4, 5.5, 6.6} }; // internally array allocated on heap
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(arr2[i][j] + "  ");
            }
            System.out.println(); // \n
        }
        System.out.println();

```

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 2, 2024, 9:51:36 AM – 9:51:36 AM) [pid: 14272]

```
1.1  2.2  3.3
4.4  5.5  6.6
```

row 0

row 1

col0 col1 col2

arr[1][2]

Writable | Smart Insert | 12 : 102 : 460

Search 9:53 AM

day05 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java | Human.java | Program02.java

Ragged Arrays

```java
        // array of arrays
        int[][] rarr = new int[4][];
        rarr[0] = new int[1];
        rarr[1] = new int[2];
        rarr[2] = new int[3];
        rarr[3] = new int[4];

        for(int i=0; i<rarr.length; i++) {
            for(int j=0; j<rarr[i].length; j++) {
                System.out.print(rarr[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 2, 2024, 9:59:31 AM – 9:59:31 AM) [pid: 9604]

```
0
0       0
0       0       0
0       0       0       0
```

Search 9:59 AM

day05 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java | Human.java | Program02.java

```java
        // array of arrays = ragged arrays
        int[][] rarr = new int[4][];
        rarr[0] = new int[1];
        rarr[1] = new int[2];
        rarr[2] = new int[3];
        rarr[3] = new int[4];

        int var = 1;
        for(int i=0; i<rarr.length; i++) {
            for(int j=0; j<rarr[i].length; j++) {
                rarr[i][j] = var;
                var++;
            }
        }

        for(int i=0; i<rarr.length; i++) {
```

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 2, 2024, 10:07:54 AM – 10:07:54 AM) [pid: 5728]

```
1
2       3
4       5       6
7       8       9       10
```

Writable | Smart Insert | 21 : 43 : 679

Search | 10:08 AM

day05 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Expl...

- > demo01 [CJ-H-02 m
  - JRE System Library
  - > src
    - > com.sunbeam
      - Human.java
      - Program01.jav
- > demo02 [CJ-H-02 m
  - JRE System Library
  - > src
    - > com.sunbeam
      - Program02.jav
- > demo03 [CJ-H-02 m
  - JRE System Library
  - > src
    - > com.sunbeam
      - Program03.jav

Program03.java

```java
package com.sunbeam;

public class Program03 {
    // method overloading -- same name but different arguments
    public static void sum(int a, int b) {
        int r = a + b;
        System.out.println("Sum: " + r);
    }
    public static void sum(int a, int b, int c) {
        int r = a + b + c;
        System.out.println("Sum: " + r);
    }
    public static void sum(int a, int b, int c, int d) {
        int r = a + b + c + d;
        System.out.println("Sum: " + r);
    }
```

Different arguments:

1. Order of args
2. Type of args
3. Count of args **

Compiler decides which method to call, depending on args passed.

Problems Javadoc Declaration Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 2, 2024, 10:12:37 AM –

```
Sum: 25
Sum: 60
Sum: 10
```

sum(12, 13);

sum(10, 20, 30);

sum(1, 2, 3, 4);

Writable | Smart Insert | 4 : 63 : 109

Search 10:13 AM

day05 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Expl... | Program03.java

- demo01 [CJ-H-02 m
  - JRE System Library
  - src
    - com.sunbeam
      - Human.java
      - Program01.ja
- demo02 [CJ-H-02 m
  - JRE System Library
  - src
    - com.sunbeam
      - Program02.ja
- demo03 [CJ-H-02 m
  - JRE System Library
  - src
    - com.sunbeam
      - Program03.ja

```java
package com.sunbeam;

public class Program03 {
    public static int divide(int a, int b) {
        int r = a / b;
        return r;
    }

    public static double divide(int a, int b) {
        double r = (double)a / b;
        return r;
    }

    public static void main(String[] args) {
        int r1 = divide(22, 7);

        double r2 = divide(22, 7);

?>      divide(22, 7);
        // collecting return value is not mandatory
    }
}
```

here compiler will not be able to decide, which method is called.

to avoid this ambiguity, compiler doesn't consider return type for fn overload

Writable | Smart Insert | 20 : 52 : 397

Search | 10:22 AM

day05 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Expl... | Program03.java

- > demo01 [CJ-H-02 m
  - JRE System Library [
  - > src
    - > com.sunbeam
      - Human.java
      - Program01.ja
- > demo02 [CJ-H-02 m
  - JRE System Library [
  - > src
    - > com.sunbeam
      - Program02.ja
- > demo03 [CJ-H-02 m
  - JRE System Library [
  - > src
    - > com.sunbeam
      - Program03.ja

```java
public class Program03 {
    public static double arraySum(double[] arr) {
        double total = 0.0;
        for(int i=0; i<arr.length; i++)
            total = total + arr[i];
        return total;
    }
    public static void main(String[] args) {
        double[] a1 = { 1.2, 1.3 };
        double r1 = arraySum(a1);
        System.out.println("Sum : " + r1);

        double[] a2 = { 1.0, 2.0, 3.0 };
        double r2 = arraySum(a2);
        System.out.println("Sum : " + r2);

        double r3 = arraySum(new double[] { 1.1, 2.2, 3.3, 4.4 });
        System.out.println("Sum : " + r3);
    }
}

/*
public class Program03 {
```

named array of 2 eles

named array of 3 eles

anonymous array of 4 eles

Writable | Smart Insert | 10 : 45 : 246

Search | 10:29 AM

day05 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Expl... | Program03.java

- demo01 [CJ-H-02 m
  - JRE System Library
  - src
    - com.sunbeam
      - Human.java
      - Program01.ja
- demo02 [CJ-H-02 m
  - JRE System Library
  - src
    - com.sunbeam
      - Program02.ja
- demo03 [CJ-H-02 m
  - JRE System Library
  - src
    - com.sunbeam
      - Program03.ja

Variable Arity Method -- since Java 5.0

```java
package com.sunbeam;

public class Program03 {
    public static double arraySum(double... arr) {
        double total = 0.0;
        for(int i=0; i<arr.length; i++)
            total = total + arr[i];
        return total;
    }
    public static void main(String[] args) {
        double r1 = arraySum(1.2, 1.3);
        System.out.println("Sum : " + r1);

        double r2 = arraySum(1.0, 2.0, 3.0);
        System.out.println("Sum : " + r2);

        double r3 = arraySum(1.1, 2.2, 3.3, 4.4);
        System.out.println("Sum : " + r3);
    }
}


/*
```

internally -- double[]

- calling method may pass array or individual args separated by ,

internally compiler convert it into an array and then pass to method.

Problems | Javadoc | Declaration | Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.REL

```
Sum : 2.5
Sum : 6.0
Sum : 11.0
```

Search 10:32 AM

day05 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java

```java
        for(int i=0; i<arr.length; i++)
            total = total + arr[i];
        return total;
    }

    public static void arrayPrint(Object... arr) {
        for(int i=0; i<arr.length; i++)
            System.out.println(arr[i]);
    }

    public static void main(String[] args) {
        double r1 = arraySum(1.2, 1.3);
        System.out.println("Sum : " + r1);

        double r2 = arraySum(1.0, 2.0, 3.0);
        System.out.println("Sum : " + r2);

        double r3 = arraySum(1.1, 2.2, 3.3, 4.4);
        System.out.println("Sum : " + r3);

        arrayPrint(11, 3.14, "Hello");
    }
}
```

In Java, each class is directly or indirectly inherited from java.lang.Object class.
So Object class reference can keep address of object of any type.

Writable | Smart Insert | 11 : 5 : 210

Search 10:40 AM

day05 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
package com.sunbeam;

public class Program04 {
    public static void swap(int x, int y) {
        int t = x;
        x = y;
        y = t;
        System.out.println("x = " + x + ", y = " + y); // x = 20, y = 10
    }

    public static void main(String[] args) {
        int a = 10, b = 20;
        System.out.println("a = " + a + ", b = " + b); // a = 10, b = 20
        swap(a, b);
        System.out.println("a = " + a + ", b = " + b); // a = 10, b = 20
    }

}
```

Calll By Value -- copy of arg is created.

When changed in fn, changes not visible in calling fn.

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.REL

```
a = 10, b = 20
x = 20, y = 10
a = 10, b = 20
```

x: 20

y: 10

local var destroy after fn

a: 10

b: 20

Search 10:53 AM

day05 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
package com.sunbeam;

public class Program04 {
    public static void swap(int x[]) {
        int t = x[0];
        x[0] = x[1];
        x[1] = t;
        System.out.println("x[0] = " + x[0] + ", x[1] = " + x[1]); // x[0] = 20, x[1] = 10
    }
    public static void main(String[] args) {
        int a[] = { 10, 20 };
        System.out.println("a[0] = " + a[0] + ", a[1] = " + a[1]); // a[0] = 10, a[1] = 20
        swap(a);
        System.out.println("a[0] = " + a[0] + ", a[1] = " + a[1]); // a[0] = 20, a[1] = 10
    }
}

/*
public class Program04 {
    public static void swap(int x, int y) {
        int t = x;
        x = y;
        y = t;
```

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.REL

```
a[0] = 10, a[1] = 20
x[0] = 20, x[1] = 10
a[0] = 20, a[1] = 10
```

In Java, primitive types are always passed by value and non-primitive types always passed by reference.

We can simulate passing primitive by reference using arrays.
(Changes done in called fn will be visible in calling fn).

Search 11:01 AM

day05 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java

```java
package com.sunbeam;

// Calculate sum of all numbers given on command-line.
public class Program05 {
    public static void main(String[] args) {
        int total = 0;
        for(int i=0; i<args.length; i++) {
            String arg = args[i];
            int num = Integer.parseInt(arg);
            total = total + num;
        }
        System.out.println("Total: " + total);
    }
}

/*
Execute on command-line:
cmd> java com.sunbeam.Program05 11 22 33 44
*/
```

Windows PowerShell | Command Prompt

+cmd line args = additional info passed on cmdline while executing the program

```
C:\Nilesh\onlinecourses\CJ-H-02\day05\demo05\bin>java com.sunbeam.Program05 11 22 33 44
Total: 110
```

Search 11:39 AM

day05 - demo07/src/com/sunbeam/Program07.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program07.java

```java
class MyClass {
    private int num1 = 1111; // field initializer
    private int num2;
    private int num3;
    private int num4 = 1; // field initializer

    {   // object/instance initializer -- since Java 5.0
        this.num2 = 111;
        System.out.println("initializer block 1");
    }

    {   // object/instance initializer -- since Java 5.0
        this.num4 = 2;
        System.out.println("initializer block 2");
    }

    {   // object/instance initializer -- since Java 5.0
        System.out.println("initializer block 3");
    }

    // constructor
    public MyClass() {
        this.num3 = 11;
        this.num4 = 3;
        System.out.println("constructor");
    }
```

Problems Javadoc Declaration Console

<terminated> Program07 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\

```
initializer block 1
initializer block 2
initializer block 3
constructor
num1=1111, num2=111, num3=11, num4=3
```

Object's fields can be initialized using.

1. Field initializers
2. Object/Instance initializers
3. Constructors

They executed in the order as given above and The next component will overwrite value initailized by previous component.

If multiple obj initializer blocks are written, they will be executed in order of their declaration in the class.

Search 12:08 PM

Association

Inheritance

All members of parent class are inherited to the child class.

Parent/Super class : generalized

inheritance

Child/Sub class : Specialized

Core Java

Arrays

2-D/Multi-dimensional array

```
double[][] arr = new double[2][3];

double[][] arr = new double[][]{ { 1.1, 2.2, 3.3 }, { 4.4, 5.5, 6.6 } };

double[][] arr = { { 1.1, 2.2, 3.3 }, { 4.4, 5.5, 6.6 } };
```

- Internally 2-D arrays are array of 1-D arrays. "arr" is array of 2 elements, in which each element is 1-D array of 3 doubles.
- Individual element is accesses as `arr[i][j]`.

Ragged array

- Ragged array is array of 1-D arrays. Each 1-D array in the ragged array may have different length.

```
int[][] arr = new int[4][];
arr[0] = new int[] { 11 };
arr[1] = new int[] { 22, 33 };
arr[2] = new int[] { 44, 55, 66 };
arr[3] = new int[] { 77, 88, 99, 110 };
for(int i=0; i<arr.length; i++) {
    for(int j=0; j<arr[i].length; j++) {
        System.out.print(arr[i][j] + ", ");
    }
}
```

Variable Arity Method

- Methods with variable number of arguments. These arguments are represented by ... and internally collected into an array.

```
public static int sum(int... arr) {
    int total = 0;
    for(int num: arr)
        total = total + num;
    return total;
}
public static void main(String[] args) {
    int result1 = sum(10, 20);
    System.out.println("Result: " + result1);
```

```
    int result2 = sum(11, 22, 33);
    System.out.println("Result: " + result2);
}
```

- If method argument is `Object... args`, it can take variable arguments of any type.
- Pre-defined methods with variable number of arguments.
  - PrintStream class: PrintStream printf(String format, Object... args);
  - String class: static String format(String format, Object... args);

Method overloading

- Methods with same name and different arguments in same scope - Method overloading.
- Arguments must differ in one of the follows
  - Count

    ```
    static int multiply(int a, int b) {
        return a * b;
    }
    static int multiply(int a, int b, int c) {
        return a * b * c;
    }
    ```

  - Type

    ```
    static int square(int x) {
        return x * x;
    }
    static double square(double x) {
        return x * x;
    }
    ```

  - Order

    ```
    static double divide(int a, double b) {
        return a / b;
    }
    static double divide(double a, int b) {
        return a / b;
    }
    ```

- Constructors have same name (as of class name) and different arguments. This is referred as "Constructor overloading".
- Note that return type is NOT considered in method overloading. Following code cause error.

```
static int divide(int a, int b) {
    return a / b;
}
static double divide(int a, int b) {
    return (double)a / b;
}
```

```
int result1 = divide(22, 7);
double result2 = divide(22, 7);
// collecting return value is not mandetory
divide(22, 7);
```

Method arguments

- In Java, primitive values are passed by value and objects are passed by reference.
- Pass by reference -- Stores address of the object. Changes done in called method are available in calling method.

```
public static void testMethod(Human h) {
    h.setHeight(5.7);
}
public static void main(String[] args) {
    Human obj = new Human(40, 76.5, 5.5);
    obj.display();  // age=40, weight=76.5, height=5.5
    testMethod(obj);
    obj.display();  // age=40, weight=76.5, height=5.7
}
```

- Pass by value -- Creates copy of the variable. Changes done in called method are not available in calling method.

```
public static void swap(int x, int y) {
    int t = x;
    x = y;
    y = t;
}
public static void main(String[] args) {
    int num1 = 11, num2 = 22;
    System.out.printf("num1=%d, num2=%d\n", num1, num2);
    swap(num1, num2);
    System.out.printf("num1=%d, num2=%d\n", num1, num2);
}
```

- Pass by reference for value/primitive types can be simulated using array.

Command line arguments

- Additional data/information passed to the program while executing it from command line -- Command line arguments.

```
terminal> java pkg.Program Arg1 Arg2 Arg3
```

- These arguments are accessible in Java application as arguments to main().

```java
package pkg;
class Program {
    public static void main(String[] args) {
        // ... args[0] = Arg1, args[1] = Arg2, args[2] = Arg3
    }
}
```

Object/Field initializer

- In C++/Java, constructor is used to initialize the fields.
- In Java, field initialization can be done using
  - Field initializer
  - Object initializer
  - Constructor
- Example:

```java
class InitializerDemo {
    int num1 = 10;
    int num2;
    int num3;

    InitializerDemo() {
        num3 = 30;
    }
    // ...

    public static void main(String[] args) {
        InitializerDemo obj = new InitializerDemo();
        System.out.printf("num1=%d, num2=%d, num3=%d\n", num1, num2, num3);
    }
}
```

final variables

- In Java, const is reserved word - but not used.
- Java has final keyword instead. It can be used for

    - final variables
    - final fields
    - final methods
    - final class
- The final local variables and fields cannot be modified after initialization.
- The final fields must be initialized any of the following.
    - Field initializer
    - Object initializer
    - Constructor
- Example:

```
class FinalDemo {
    final int num1 = 10;
    final int num2;
    final int num3;

    {
        num2 = 20;
    }

    FinalDemo() {
        num3 = 30;
    }
    public void display() {
        System.out.printf("num1=%d, num2=%d, num3=%d\n", num1, num2, num3);
    }

    public static void main(String[] args) {
        final int num4 = 40;

        final FinalDemo obj = new FinalDemo();
        obj.display();
    }
}
```

static keyword

- In OOP, static means "shared" i.e. static members belong to the class (not object) and shared by all objects of the class.
- Static members are called as "class members"; whereas non-static members are called as "instance memebers".
- In Java, static keyword is used for
    - static fields
    - static methods
    - static block
    - static import
- Note that, static local variables cannot be created in Java.

Static fields

- Copies of non-static/instance fields are created one for each object.
- Single copy of the static/class field is created (in method area) and is shared by all objects of the class.
- Can be initialized by static field initializer or static block.
- Accessible in static as well as non-static methods of the class.
- Can be accessed by class name or object name outside the class (if not private). However, accessing via object name is misleading (avoid it).

Static methods

- Methods can be called from outside the class (if not private) using class name or object name. However, accessing via object name is misleading (avoid it).
- When needs to call a method without object, then make it static.
- Since static methods are designed to be called on class name, they do not have "this" reference. Hence, cannot access non-static members in the static method (directly). However, we can access them on an object reference.
- Applications
    - To initialize/access static fields.
    - Helper/utility methods

    ```
    import java.util.Arrays;
    // in main()
    int[] arr = { 33, 88, 44, 22, 66 };
    Arrays.sort(arr);
    System.out.println(Arrays.toString(arr));
    ```

    - Factory method - to create object of the class

    ```
    import java.util.Calendar;
    // in main()
    //Calendar obj = new Calendar(); // compiler error
    Calendar obj = Calendar.getInstance();
    System.out.println(obj);
    ```

Static field initializer

- Similar to field initializer, static fields can be initialized at declaration.

```
// static field
static double price = 5000.0;
```

static keyword

Static Method

- If we want to access non static members of the class then we should define non static method inside class.

```
class Test{
    private int num1;
    public int getNum1( ){
        return this.num1;
    }
    public void setNum1( int num1 ){
        this.num1 = num1;
    }
}
```

- If we want to access static members of the class then we should define static method inside class.

```
class Test{
    private static int num2;
    public static int getNum2( ){
        return Test.num2;
    }
    public void setNum2( int num2 ){
        Test.num2 = num2;
    }
}
```

Why static method do not get this reference?

```
- If we call non static method on instance then method gets this reference.
- Static method is designed to call on class name.
- Since static method is not designed to call on instance, it doesn't get this
reference.
```

- Since static method do not get this reference, we can not access non static members inside static method.
- In other words, static method can access only static members of the class directly.
- If we want to access non static members inside static method then we need to use instance of the class.

```
class Program{
    int num1 = 10;
    static int num2 = 20;
    public static void main( String[] args ){
        //System.out.println("Num1    :   "+num1);    //Compiler error
        Program p = new Program( );
        System.out.println("Num1    :   "+p.num1);    //OK: 10
        System.out.println("Num1    :   "+new Program().num1);    //OK: 10
```

```
        System.out.println("Num2    :   "+num2);    //OK: 20
        System.out.println("Num2    :   "+Program.num2);    //OK:20
    }
}
```

- Inside non static method, we can access static as well as non static members directly.

```
class Test{
    private int num1 = 10;
    private static int num2 = 20;
    public void printRecord( ){
        System.out.println("Num1    :   "+this.num1);   //OK
        System.out.println("Num2    :   "+Test.num2);   //OK
    }
}
```

- Inside method, if there is a need to use this reference then we should declare method non static otherwise we should declare method static.

```
class Math{
    public static int power( int base, int index ){
        int result = 1;
        for( int count = 1; count <= index; ++ count ){
            result = result * base;
        }
        return result;
    }
}
class Program{
    public static void main(String[] args) {
        int result = Math.power(2, 3);
        System.out.println("Result  :   "+result);
    }
}
```

- Method local variable get space once per method call.
- We can declare, method local variable final but we can not declare it static.
    - static variable is also called as class level variable.
    - class level variables should exist at class scope.
    - Hence we can not declare local variable static. But we can declare field static.

```
class Program{
    private static int number;  //OK
    public static void print( ){
        //static int number = 0;  //Not OK
        number = number + 1;
```

```
        System.out.println("Number  :   "+number);
    }
    public static void main(String[] args) {
        Program.print();    //1
        Program.print();    //2
        Program.print();    //3
    }
}
```

Static block

- Like Object/Instance initializer block, a class can have any number of static initialization blocks, and they can appear anywhere in the class body.
- Static initialization blocks are executed in the order their declaration in the class.
- A static block is executed only once when a class is loaded in JVM.
- Example:

```
class Program {
    static int field1 = 10;
    static int field2;
    static int field3;
    static final int field4;

    static {
        // static fields initialization
        field2 = 20;
        field3 = 30;
    }

    static {
        // initialization code
        field4 = 40;
    }
}
```

Static import

- To access static members of a class in the same class, the "ClassName." is optional.
- To access static members of another class, the "ClassName." is mandetory.
- If need to access static members of other class frequently, use "import static" so that we can access static members of other class direcly (without ClassName.).

```
import static java.lang.Math.*;
class Program {
   public static double calcArea(double rad) {
    return Math.PI * rad * rad;
}
}
```

Singleton class

- Design patterns are standard solutions to the well-known problems.
- Singleton is a design pattern.
- It enables access to an object throughout the application source code.
- Singleton class is a class whose single object is created throughout the application.
- To make a singleton class in Java
  - step 1: Write a class with desired fields and methods.
  - step 2: Make constructor(s) private.
  - step 3: Add a private static field to hold instance of the class.
  - step 4: Initialize the field to single object using static field initializer or static block.
  - step 5: Add a public static method to return the object.
- Code:

```java
public class Singleton {
    // fields and methods
    // since ctor is declared private, object of the class cannot be created
outside the class.
    private Singleton() {
        // initialization code
    }
    // holds reference of "the" created object.
    private static Singleton obj;
    static {
        // as static block is executed once, only one object is created
        obj = new Singleton();
    }
    // static getter method so that users can access the object
    public static Singleton getInstance() {
        return obj;
    }
}
```

```java
class Program {
    public static void testMethod() {
        Singleton obj2 = Singleton.getInstance();
        // ...
    }

    public static void main(String[] args) {
        Singleton obj1 = Singleton.getInstance();
        // ...
```

```
    }
}
```

Association

- If "has-a" relationship exist between the types, then use association.
- To implement association, we should declare instance/collection of inner class as a field inside another class.
- There are two types of associations
  - Composition
  - Aggregation
- Example 1:

```
public class Engine {
    // ...
}
```

```
public class Person {
    private String name;
    private int age;
    // ...
}
```

```
public class Car {
    private Engine engine;
    private Person driver;
    // ...
}
```

- Example 2:

```
public class Wall {
    // ...
}
```

```
public class Person {
    // ...
}
```

```java
public class Classroom {
    private Wall[] walls = new Wall[4];
    private ArrayList<Person> students = new ArrayList<>();
    // ...
}
```

Composition

- Represents part-of relation i.e. tight coupling between the objects.
- The inner object is essential part of outer object.
    - Engine is part of Car.
    - Wall is part of ClassRoom

Aggregation

- Represents has-a relation i.e. loose coupling between the objects.
- The inner object can be added, removed, or replaced easily in outer object.
    - Car has a Driver.
    - Company has Employees.

Inheritance

- If "is-a"/"kind-of" relationship exist between the types, then use inheritance.
- Inheritance is process -- generalization to specialization.
- All members of parent class are inherited to the child class.
- Example:
    - Manager is a Employee
    - Mango is a Fruit
    - Triangle is a Shape
- In Java, inheritance is done using extends keyword.

```java
class SubClass extends SuperClass {
    // ...
}
```

- Java doesn't support multiple implementation inheritance i.e. a class cannot be inherited from multiple super-classes.
- However Java does support multiple interface inheritance i.e. a class can be inherited from multiple super interfaces.

super keyword

- In sub-class, super-class members are referred using "super" keyword.

- Calling super class constructor

- By default, when sub-class object is created, first super-class constructor (param-less) is executed and then sub-class constructor is executed.
- "super" keyword is used to explicitly call super-class constructor.

```
class Person {
    // ...
    public Person(String name, int age) {
        // ...
    }
}
class Student extends Person {
    // ...
    public Student(String name, int age, int roll, double marks) {
        super(name, age); // calls parameterized ctor of super class -- must
be first line only
        // ...
    }
}
```

- Accessing super class members
  - Super class members (non-private) are accessible in sub-class directly or using "this" reference. These members can also be accessed using "super" keyword.
  - However, if sub-class method signature is same as super-class signature, it hides/shadows method of the super class i.e. super-class method is not directly visible in sub-class.
  - The "super" keyword is mandetory for accessing such hidden members of the super-class.

```
class Person {
    // ...
    public String getName() {
        // ...
    }
    public int getAge() {
        // ...
    }
    public void display() {
        // display name and age
    }
}
class Student extends Person {
    // ...
    public void display() {
        System.out.println(this.getName()); // getName() is inherited from
super-class
        System.out.println(getAge()); // getAge() is inherited from super-
class
        super.display(); // Person.display() is hidden due to
Student.display()
            // must use super keyword to call hidden method of super class.
        // display roll and marks
```

```
    }
}
```

Inheritance

Types of inheritances

- Single inheritance

```
class A {
    // ...
}
class B extends A {
    // ...
}
```

- Multiple inheritance

```
class A {
    // ...
}
class B {
    // ...
}
class C extends A, B // not allowed in Java
{
    // ...
}
```

```
interface A {
    // ...
}
interface B {
    // ...
}
class C implements A, B // allowed in Java
{
    // ...
}
```

- Hierarchial inheritance

```
class A {
    // ...
}
```

```
class B extends A {
    // ...
}
class C extends A {
    // ...
}
```

- Multi-level inheritance

```
class A {
    // ...
}
class B extends A {
    // ...
}
class C extends B {
    // ...
}
```

- Hybrid inheritance: Any combination of above types

Up-casting & Down-casting

- Up-casting: Assigning sub-class reference to a super-class reference.
    - Sub-class "is a" Super-class, so no explicit casting is required.
    - Using such super-class reference, super-class methods overridden into sub-class can also be called.

```
Employee e = new Employee();
Person p = e; // up-casting
p.setName("Nilesh");    // okay - calls Person.setName().
p.setSalary(30000.0);   // error
p.display();    // calls overridden Employee.display().
```

- Down-casting: Assigning super-class reference to sub-class reference.
    - Every super-class is not necessarily a sub-class, so explicit casting is required.

```
Person p1 = new Employee();
Employee e1 = (Employee)p1; // down-casting - okay - Employee reference will
point to Employee object
```

```
Person p2 = new Person();
Employee e2 = (Employee)p2; // down-casting - ClassCastException - Employee
```

```
reference will point to Person object
```

p
Person
Upcasting
Employee
emp
"Sandeep"
String
39
int
378
int
45000.50
float
name
age
empid
salary
[Employee Instance]
Instance of Person class which is
Exist inside Employee class.

The instance, in which space is reserved for name & age and we can call only showRecord method on it is an instance of Person class.

p
Pereson
Downcasting
Employee
emp
"Sandeep"
String
39
int
378
int
45000.50
float
name
age
empid
salary
[Employee Instance]
Instance of Person class which is
Exist inside Employee class.

day05 - demo07/src/com/sunbeam/Program07.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program07.java

```java
class MyClass {
    private int num1 = 1111; // field initializer
    private int num2;
    private int num3;
    private int num4 = 1; // field initializer

    {   // object/instance initializer -- since Java 5.0
        this.num2 = 111;
        System.out.println("initializer block 1");
    }

    {   // object/instance initializer -- since Java 5.0
        this.num4 = 2;
        System.out.println("initializer block 2");
    }

    {   // object/instance initializer -- since Java 5.0
        System.out.println("initializer block 3");
    }

    // constructor
    public MyClass() {
        this.num3 = 11;
        this.num4 = 3;
        System.out.println("constructor");
    }
```

Problems Javadoc Declaration Console

<terminated> Program07 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\

```
initializer block 1
initializer block 2
initializer block 3
constructor
num1=1111, num2=111, num3=11, num4=3
```

Object's fields can be initialized using.

1. Field initializers
2. Object/Instance initializers
3. Constructors

They executed in the order as given above and The next component will overwrite value initailized by previous component.

If multiple obj initializer blocks are written, they will be executed in order of their declaration in the class.

Search 12:08 PM

day05 - demo08/src/com/sunbeam/Program08.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program08.java

```java
        return 2 * PI * this.radius;
    }
}

public class Program08 {
    public static void main(String[] args) {
        final double rad; // final var declaration
        rad = 7; // final var initialization
        //rad = 14; // compiler error: once initialized, cannot be modified
        Circle c1 = new Circle(rad);
        System.out.println("Area: " + c1.calcArea());
        System.out.println("Peri: " + c1.calcPeri());

        final double rad2 = 14; // final var declaration & initialization
        final Circle c2 = new Circle(rad2); // final ref declaration & initialization
        c2.setRadius(70); // obj state can be modified (as reference is final)
        //c2 = null; // compiler error: reference once initialized, cannot be modified
        System.out.println("Area: " + c2.calcArea());
        System.out.println("Peri: " + c2.calcPeri());
    }
}
```

Writable | Smart Insert | 44 : 11 : 1490

Search 12:31 PM

day05 - demo09/src/com/sunbeam/Program09.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program09.java

```java
class Chair {
    private int height, weight;
    private static int price = 100;
    public Chair() {
        this.height = 0;
        this.weight = 0;
    }
    public Chair(int height, int weight) {
        this.height = height;
        this.weight = weight;
    }
    public void display() {
        System.out.printf("height: %d, weight: %d, ",
                this.height, this.weight);
        //System.out.printf("price: %d\n", this.price); // accessible, but misleading
        System.out.printf("price: %d\n", Chair.price); // more readable
    }
    public static void setPrice(int price) {
        Chair.price = price;
    }
    public static int getPrice() {
        return Chair.price;
    }
```

The static members should be accessed using class name (better readability).

ClassName.fieldName = value;
ClassName.methodName(...);

The static members if declared private, cannot be directly accessed outside the class.

Typically static methods are used to operate on static fields.

Writable | Smart Insert | 23 : 35 : 624

Search 1:08 PM

day06 - demo01/src/com/sunbeam/Chair.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Chair.java

```java
public class Chair {
    private int height;
    private int weight;
    // static field initializer
    private static double price = 100;

    static { // static block
        price = 200;
        System.out.println("static block 1");
    }
    static { // static block
        price = 300;
        System.out.println("static block 2");
    }

    public Chair() {
        this(0, 0); // constructor chaining
    }
    public Chair(int height, int weight) {
        this.height = height;
        this.weight = weight;
    }
```

Program01.java

```java
package com.sunbeam;

public class Program01 {
    public static void main(String[] args) {
        System.out.println("Chair price: "
                           + Chair.getPrice());
    }
}
```

Applications of static block:

1. Initialize static fields of the class
2. One time initialization for the whole class

The static blocks are executed only once when class is loaded for first time into JVM.

If class have multiple static blocks, they will be executed in order of their declaration in class.

Writable | Smart Insert | 13 : 29 : 274

Search 9:42 AM

day06 - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
- demo02 [CJ-H-02 main]
- demo03 [CJ-H-02 main]
  - JRE System Library [JavaSE
  - src

UML diagram
- class diagram
  - to represent classes and their relations

has-a reln

is-a reln

Date ◇— Emp

Date
- day: int - month: int - year: int
+ Date() + Date(int,int,int) + accept():void + display():void + getDay():int + setDay(int):void + ...

has-a

Emp
- id: int - name: String - salary: double - joining: Date
+ Emp() + Emp(int,String,double,Date) + accept():void + display():void + getJoining():Date + setJoining(Date):void + ...

- private
+ public
# protect
* default

d1: Date —has-a◇— e1: Emp

Object Diagram

src - demo03

Search

11:26 AM

day06 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Date.java | Emp.java | Program03.java

```java
package com.sunbeam;

public class Program03 {
    public static void main(String[] args) {
        Emp e1 = new Emp();
        e1.display();
        System.out.println();

        Date d2 = new Date(1, 5, 2004);
        Emp e2 = new Emp(3, "Nilesh Ghule", 50000.0, d2);
        e2.display();
        System.out.println();

        Emp e3 = new Emp();
        e3.accept();
        e3.display();
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts

```
Id: 0
Name:
Sal: 0.000000
Joining Date: 1-1-2000

Id: 3
Name: Nilesh Ghule
Sal: 50000.000000
Joining Date: 1-5-2004

Id: 7
Name: James Bond
Sal: 80000
Joining Date (day month yea
Id: 7
Name: James Bond
Sal: 80000.000000
Joining Date: 1-1-1900
```

Search 12:00 PM

day06 - Spring Tool Suite 4

File Edit Navigate Search Project Run Window Help

Search 12:13 PM

day06 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Person.java

```java
package com.sunbeam;

public class Person {
    private String name;
    private int age;
    public Person() {
        this.name = "";
        this.age = 1;
        System.out.println("Person() called");
    }
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
        System.out.println("Person(String,int) called");
    }
```

Program04.java

```java
package com.sunbeam;

public class Program04 {
    public static void main(String[] args) {
        //Student s1 = new Student();
        Student s2 = new Student("Nilesh", 20, 424, "DAC
    }
}
```

Student.java

```java
public class Student extends Person {
    private int roll;
    private String course;
    private int marks;

    public Student() {
        this.roll = 1;
        this.course = "";
        this.marks = 0;
        System.out.println("Student() called");
    }
    public Student(String name, int age, int roll, Strin
        //this.name = name; // error: private members of
        super(name, age); // invokes super class constru
        this.roll = roll;
        this.course = course;
        this.marks = marks;
        System.out.println("Student(String,int,int,Strin
    }

    public int getRoll() {
        return roll;
    }
    public void setRoll(int roll) {
        this.roll = roll;
```

Writable | Smart Insert | 5 : 38 : 124

Search 12:35 PM

day06 - demo04/src/com/sunbeam/Student.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Person.java | Program04.java

```java
        System.out.println("Person() called");
    }
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
        System.out.println("Person(String,int)
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    public void display() {
        System.out.printf("Name: %s\nAge: %d\n
    }
```

Student.java

```java
        return course;
    }
    public void setCourse(String course) {
        this.course = course;
    }
    public int getMarks() {
        return marks;
    }
    public void setMarks(int marks) {
        this.marks = marks;
    }

    public void display() {
        //// getName() and getAge() inherited fr
        System.out.printf("Name: %s\nAge: %d\n",
            this.getName(), this.getAge());
        //// display() is also inherited from Pe
        //this.display();
        System.out.printf("Roll: %d\nCourse: %s\
                this.roll, this.course, this.mar
    }
```

Writable | Smart Insert | 47 : 11 : 1229

Search 12:52 PM

day06 - demo04/src/com/sunbeam/Person.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

"super" keyword is used to access super-class members (non-private) from sub-class methods.

Person.java × Program04.java

```java
        this.name = name;
        this.age = age;
        System.out.println("Person(String,int)
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    public void display() {
        System.out.printf("Name: %s\nAge: %d\n
    }
}
```

if super-cls method name is same as sub-cls method name, then "super" keyword is mandatory to access the super class method.

Student.java ×

```java
        return course;
    }
    public void setCourse(String course) {
        this.course = course;
    }
    public int getMarks() {
        return marks;
    }
    public void setMarks(int marks) {
        this.marks = marks;
    }
    public void display() {
        //// getName() and getAge() inherited
        //System.out.printf("Name: %s\nAge: %d
            //this.getName(), this.getAge());
        //// display() is also inherited from
        super.display();
        System.out.printf("Roll: %d\nCourse: %
                this.roll, this.course, this.m
    }
}
```

if super class method names are not same as sub-cls method names, you can use "super" or "this".

Writable | Smart Insert | 24 : 6 : 478

Search 12:57 PM

Inheritance

To avoid ambiguity errors due to multiple inheritance, Java lanaguage doesn't have support for multiple implementation(class) inheritance. However, Java allows multiple interface inheriance.

All members of parent class are inherited to the child class.

Parent/Super class : generalized

inheritance

Child/Sub class : Specialized

super keyword:

1. invoke super class ctor from sub class ctor.
2. access super class members from sub class non-static methods.

```
class Person {
    =
    public void display() {
        =
    }
}
class Student extends Person {
    =
    public void display() {
        super.display();
        =
    }
}
main():
    Student s1 = new Student();
    s1.display();
```

// cannot call Person.display on Student ref → Method shadowing

In C++:

```
class Programmer {
    public:
        void display() { ... }
};
class Teacher {
    public:
        void display() { ... }
};
class ProgrammingTeacher:
    public Programmer, Teacher {
    // ...
};
main():
ProgrammingTeacher obj;
obj.display(); // ambiguity error
```

Core Java

Inheritance

Types of inheritances

- Single inheritance

```
class A {
    // ...
}
class B extends A {
    // ...
}
```

- Multiple inheritance

```
class A {
    // ...
}
class B {
    // ...
}
class C extends A, B // not allowed in Java
{
    // ...
}
```

```
interface A {
    // ...
}
interface B {
    // ...
}
class C implements A, B // allowed in Java
{
    // ...
}
```

- Hierarchial inheritance

```
class A {
    // ...
}
```

```
class B extends A {
    // ...
}
class C extends A {
    // ...
}
```

- Multi-level inheritance

```
class A {
    // ...
}
class B extends A {
    // ...
}
class C extends B {
    // ...
}
```

- Hybrid inheritance: Any combination of above types

Up-casting & Down-casting

- Up-casting: Assigning sub-class reference to a super-class reference.
    - Sub-class "is a" Super-class, so no explicit casting is required.
    - Using such super-class reference, super-class methods overridden into sub-class can also be called.

```
Employee e = new Employee();
Person p = e; // up-casting
p.setName("Nilesh");    // okay - calls Person.setName().
p.setSalary(30000.0);   // error
p.display();    // calls overridden Employee.display().
```

- Down-casting: Assigning super-class reference to sub-class reference.
    - Every super-class is not necessarily a sub-class, so explicit casting is required.

```
Person p1 = new Employee();
Employee e1 = (Employee)p1; // down-casting - okay - Employee reference will
point to Employee object
```

```
Person p2 = new Person();
Employee e2 = (Employee)p2; // down-casting - ClassCastException - Employee
```

```
reference will point to Person object
```

instanceof operator

- Java's instanceof operator checks if given reference points to the object of given type (or its sub-class) or not. Its result is boolean.

```
Person p = new Employee();
boolean flag = p instanceof Employee; // true
```

```
Person p = new Person();
boolean flag = p instanceof Employee; // false
```

- Typically "instanceof" operator is used for type-checking before down-casting.

```
Person p = new SomeClass();
if(p instanceof Employee) {
    Employee e = (Employee)p;
    System.out.println("Salary: " + e.getSalary());
}
```

Polymorphism

- Poly=Many, Morphism=Forms
- Polymorphism is taking many forms.
- OOP has two types of Polymorphism
  - Compile-time Polymorphism / Static Polymorphism
    - Implemented by "Method overloading".
    - Compiler can identify which method to be called at compile time depending on types of arguments. This is also referred as "Early binding".
  - Run-time Polymorphism / Dynamic Polymorphism
    - Implemented by "Method overriding".
    - The method to be called is decided at runtime depending on type of object. This is also referred as "Late binding" or "Dyanmic method dispatch".
  - Process of calling method of sub class using reference of super class is called as dynamic method dispatch. ###Access Modifier
  - private (lowest)
  - default
  - protected
  - public (highest)

Method overriding

- Redefining a super-class method in sub-class with exactly same signature is called as "Method overriding".
- Programmer should override a method in sub-class in one of the following scenarios
  - Super-class has not provided method implementation at all (abstract method).
  - Super-class has provided partial method implementation and sub-class needs additional code. Here sub-class implementation may call super-class method (using super keyword).
  - Sub-class needs different implementation than that of super-class method implementation.
- Rules of method overriding in Java
  - Each method in Java can be overridden unless it is private, static or final.
  - Sub-class method must have same or wider access modifier than super-class method.

    ```
    class SuperClass {
        protected Number calculate(Integer i, Float f) {
            // ...
        }
    }
    class SubClass extends SuperClass {
        /*protected or*/ public Number calculate(Integer i, Float f) {
            // ...
        }
    }
    ```

  - Arguments of sub-class method must be same as of super-class method. The return-type of sub-class method can be same or sub-class of the super-class's method's return-type. This is called as "covarient" return-type.

    ```
    class SuperClass {
        public Number calculate(Integer i, Float f) {
            // ...
        }
    }
    class SubClass extends SuperClass {
        // Double is inherited from Numer (i.e. return-type of super-class
    method)
        public Double calculate(Integer i, Float f) {
            // ...
        }
    }
    ```

  - Checked exception list in sub-class method should be same or subset of exception list in super-class method.

    ```
    class SuperClass {
        public void testMethod() throws IOException, SQLException {
            // ...
        }
    ```

```
}
class SubClass extends SuperClass {
    public void testMethod() throws IOException {
        // ...
    }
}
```

- If these rules are not followed, compiler raises error or compiler treats sub-class method as a new method.

```
class A {
    void method1() {
    }
    void method2() {
    }
}
```

```
class B extends A {
    void method1() {
        // overridden
    }
    void method2(int x) {
        // treated as new method
    }
}
```

```
// In main()
A obj = new B();
obj.method1();  // B.method1() -- overridden
obj.method2();  // A.method2() -- method2() is not overridden
```

- Java 5.0 added @Override annotation (on sub-class method) informs compiler that programmer is intending to override the method from the super-class.
- @Override checks if sub-class method is compatible with corresponding super-class method or not (as per rules). If not compatible, it raise compile time error.
- Note that, @Override is not compulsory to override the method. But it is good practice as it improves readability and reduces human errors.

final keyword

final variables/fields

- Cannot be modified once initialized.
- Refer earlier notes.

final method

- If implementation of a super-class method is logically complete, then the method should be declared as final.
- Such final methods cannot be overridden in sub-class. Compiler raise error, if overridden.
- But final methods are inherited into sub-class i.e. The super-class final methods can be invoked in sub-class object (if accessible).

final class

- If implementation of a super-class is logically complete, then the class should be declared as final.
- The final class cannot be extended into a sub-class. Compiler raise error, if inherited.
- Effectively all methods in final class are final methods.
- Examples of final classes
    - java.lang.Integer (and all wrapper classes)
    - java.lang.String
    - java.lang.System

Object class

- Non-final and non-abstract class declared in java.lang package.
- In java, all the classes (not interfaces) are directly or indirectly extended from Object class.
- In other words, Object class is ultimate base class/super class.
- Object class is not inherited from any class or implement any interface.
- It has a default constructor.
    - Object o = new Object();

Object class methods (read docs)

- Parameter less constructor
    - public Object();
- Returns string representation of object state
    - public String toString();
- Comparing current object with another object
    - public boolean equals(Object);
- Used while storing object into set or map collections
    - public native int hashCode();
- Create shallow copy of the object
    - protected native Object clone() throws CloneNotSupportedException;
- Called by garbage collector (like C++ destructor)
    - protected void finalize() throws Throwable;
- Get metadata about the class
    - public final native Class<?> getClass();
- For thread synchronization
    - public final native void notify();
    - public final native void notifyAll();
    - public final void wait() throws InterruptedException;
    - public final native void wait(long) throws InterruptedException;

    - public final void wait(long, int) throws InterruptedException;

toString() method

- Non-final method of java.lang.Object class.
    - public String toString();
- Definition of Object.toString():

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

- To return state of Java instance in String form, programmer should override toString() method.
- The result in toString() method should be a concise, informative, and human-readable.
- It is recommended that all subclasses override this method.
- Example:

```
class Person {
    // ...
    @Override
    public String toString() {
        return "Name=" + this.name + ", Age=" + this.age;
    }
}
```

Inheritance vs Association

- Inheritance: is-a relation
    - Book is-a Product
    - Album is-a Product
    - Labor is-a Employee
    - Employee is-a Person
    - Batter is-a Player
    - ...
- Association: has-a relation
    - Employee has-a joining Date
    - Person has-a birth Date
    - Cart has Products
    - Bank has Accounts
    - ...

abstract keyword

- In Java, abstract keyword is used for
    - abstract method
    - abstract class

Fragile base class problem

- If changes are done in super-class methods (signatures), then it is necessary to modify and recompile all its sub-classes. This is called as "Fragile base class problem".
- This can be overcomed by using interfaces.

```
class A{
    public void print( ){
        //System.out.print("Hello,");
        System.out.print("Good Morning,");
    }
}
class B extends A{
    @Override
    public void print( ){
        super.print();
        System.out.println("Have a nice day!!");
    }
}
class C extends A{
    @Override
    public void print( ){
        super.print();
        System.out.println("Good day!!");
    }
}
class Program{
    public static void main(String[] args) {
        A a = null;
        a = new B();    a.print();  //Good Morning,,Have a nice day!!
        a = new C();    a.print();  //Good Morning,,Good day!!
    }
}
```

Interface (Java 7 or Earlier)

- Interfaces are used to define standards/specifications. A standard/specification is set of rules.
- Interfaces are immutable i.e. once published interface should not be modified.
- Interfaces contains only method declarations. All methods in an interface are by default abstract and public.
- They define a "contract" that is must be followed/implemented by each sub-class.

```
interface Displayable {
    public abstract void display();
}
```

```
interface Acceptable {
    abstract void accept(Scanner sc);
}
```

```
interface Shape {
    double calcArea();
    double calcPeri();
}
```

- Interfaces enables loose coupling between the classes i.e. a class need not to be tied up with another class implementation.
- Interfaces cannot be instantiated, they can only be implemented by classes or extended by other interfaces.
- Java 7 interface can only contain public abstract methods and static final fields (constants). They cannot have non-static fields, non-static methods, and constructors.
- Examples:
    - java.io.Closeable / java.io.AutoCloseable
    - java.lang.Runnable
    - java.util.Collection, java.util.List, java.util.Set, ...
- Example 1: Multiple interface inheritance is allowed.

```
interface Displayable {
    void display();
}
interface Acceptable {
    void accept();
}

class Person implements Acceptable, Displayable {
    // ...
    public void accept() {
        // ...
    }
    public void display() {
        // ...
    }
}
```

- Example 2: Interfaces can have public static final fields.

```
interface Shape {
    /*public static final*/ double PI = 3.142;

    /*public abstract*/ double calcArea();
    /*public abstract*/ double calcPeri();
```

```
}

class Circle implements Shape {
    private double radius;
    // ...
    public double calcArea() {
        return PI * this.radius * this.radius;
    }
    public double calcPeri() {
        return 2 * Shape.PI * this.radius;
    }
}
```

- Example 3: If two interfaces have same method, then it is implemented only once in sub-class.

```
interface Displayable {
    void print();
}
interface Showable {
    void print();
}
class MyClass implements Displayable, Showable {
    // ...
    public void print() {
        // ...
    }
}
class Program {
    public static void main(String[] args) {
        Displayable d = new MyClass();
        d.print();
        Showable s = new MyClass();
        s.print();
        MyClass m = new MyClass();
        m.print();
    }
}
```

Types of inheritance in OOPS

- Interface inheritance
  - Single inheritance [Allowed in Java]
  - Multiple inheritance [Allowed in Java]
  - Hierarchical inheritance [Allowed in Java]
  - Multilevel inheritance [Allowed in Java]
- Implementation inheritance
  - Single inheritance [Allowed in Java]
  - Multiple inheritance [Not Allowed in Java]
  - Hierarchical inheritance [Allowed in Java]

    - Multilevel inheritance [Allowed in Java]

- Interface syntax
    - Interface : I1, I2, I3
    - Class : C1, C2, C3
    - class C1 implements I1 // okay
    - class C1 implements I1, I2 // okay
    - interface I2 implements I1 // error
    - interface I2 extends I1 // okay
    - interface I3 extends I1, I2 // okay
    - class C2 implements C1 // error
    - class C2 extends C1 // okay
    - class C3 extends C1, C2 // error
    - interface I1 extends C1 // error
    - interface I1 implements C1 // error
    - class C2 implements I1, I2 extends C1 // error
    - class C2 extends C1 implements I1,I2 // okay

abstract method

- If implementation of a method in super-class is not possible/incomplete, then method is declared as abstract.
- Abstract method does not have definition/implementation.

```
// Employee class
abstract double calcTotalSalary();
```

- If class contains one or more abstract methods, then class must be declared as abstract. Otherwise compiler raise an error.
- The super-class abstract methods must be overridden in sub-class; otherwise sub-class should also be marked abstract.
- The abstract methods are forced to be implemented in sub-class. It ensures that sub-class will have corresponding functionality.
- The abstract method cannot be private, final, or static.
- Example: abstract methods declared in Number class are:
    - abstract int intValue();
    - abstract float floatValue();
    - abstract double doubleValue();
    - abstract long longValue();

abstract class

- If implementation of a class is logically incomplete, then the class should be declared abstract.
- If class contains one or more abstract methods, then class must be declared as abstract.
- An abstract class can have zero or more abstract methods.
- Abstract class object cannot be created; however its reference can be created.

- Abstract class can have fields, methods, and constructor.
- Its constructor is called when sub-class object is created and initializes its (abstract class) fields.
- If object of a class is not logical (corresponds to real-world entity), then class can be declared as abstract.
- Example:
    - java.lang.Number
    - java.lang.Enum

class vs abstract class vs interface

- class
    - Has fields, constructors, and methods
    - Can be used standalone -- create objects and invoke methods
    - Reused in sub-classes -- inheritance
    - Can invoke overridden methods in sub-class using super-class reference -- runtime polymorphism
- abstract class
    - Has fields, constructors, and methods
    - Cannot be used independently -- can't create object
    - Reused in sub-classes -- inheritance -- Inherited into sub-class and must override abstract methods
    - Can invoke overridden methods in sub-class using super-class reference -- runtime polymorphism
- interface
    - Has only method declarations
    - Cannot be used independently -- can't create object
    - Doesn't contain anything for reusing (except static final fields)
    - Used as contract/specification -- Inherited into sub-class and must override all methods
    - Can invoke overridden methods in sub-class using super-class reference -- runtime polymorphism
    - Java support multiple interface inheritance

What is the difference between abstract class and interface? / When we should use abstract class and interface?

Abstract class

```java
abstract class Shape{
    public abstract void calculateArea( );
}
class Rectangle extends Shape{
    @Override
    public void calculateArea(){
        //TODO
    }
}
class Circle extends Shape{
    @Override
```

```
    public void calculateArea(){
        //TODO
    }
}
class Triangle extends Shape{
    @Override
    public void calculateArea(){
        //TODO
    }
}
```

- If "is-a" relationship is exist between super type & sub type and if we want to maintain same method signature/design in all the sub clases then we should declare super type abstract.

```
Shape[] arr = new Shape[ 3 ];
arr[ 0 ] = new Rectangle();
arr[ 1 ] = new Circle();
arr[ 2 ]  = new Triangle();
```

- Using abstract class, we can group instances of related type together.
- Abstract class can extend only one abstract class / concrete class. In other words, using abstract class, we can not achieve multiple inheritance.
- We can define constructor inside abstract class.
- Abstract class may / may not contain abstract method.
- In General, if state is involved in super type then super type should be abstract class.

Interface

```
interface Printable{
    void printRecord( );
}
class Complex implements Printable{
    @Override
    public void printRecord(){
        System.out.println("Print complex number");
    }
}
class Point implements Printable{
    @Override
    public void printRecord(){
        System.out.println("Print point");
    }
}
class Date implements Printable{
    @Override
    public void printRecord(){
        System.out.println("Print date");
```

```
    }
}
```

- If "is-a" relationship is not exist(can-do relationship is exist) between super type & sub type and if we want to maintain same method design in all the sub classes then we should declare super type interface.

```
Printable[] arr = new Printable[ 3 ];
arr[ 0 ] = new Complex();
arr[ 1 ] = new Point();
arr[ 2 ] = new Date();
```

- Using interface, we can group instances of unrelated type together.
- Interface can extend more than one interfaces. In other words, using interface, we can achive multiple inheritance.
- We can not define constructor inside interface.
- Interface methods are by default abstract.
- In General, if state is not involved in super type then super type should be interface.

equals() method

- Non-final method of java.lang.Object class.
    - public boolean equals(Object other);
- Definition of Object.equals():

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- To compare the object contents/state, programmer should override equals() method.
- This equals() must have following properties:
    - Reflexive: for any non-null reference value x, x.equals(x) should return true.
    - Symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
    - Transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
    - Consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
    - For any non-null reference value x, x.equals(null) should return false.
- Example:

```
class Employee {
    // ...
```

```java
    @Override
    public boolean equals(Object obj) {
        if(obj == null)
            return false;

        if(this == obj)
            return true;

        if(! (obj instanceof Employee))
            return false;

        Employee other = (Employee) obj;
        if(this.id == other.id)
            return true;
        return false;
    }
}
```

equals() overriding

Rules for overriding equals()

① equals() must be reflexive.
d1.equals(d1) must be true.

② equals() must be symmetric.
if d1.equals(d2) is true,
then d2.equals(d1) must be true.

③ equals() must be transitive.
if d1.equals(d2) is true and
d2.equals(d3) is true, then
d1.equals(d3) must be true.

④ equals() must be consistent
Result of d1.equals(d2) should
remain same until any
of the object state modified.

⑤ null comparison
d1.equals(null) must be false.

⑥ Comparison with incompatible type.
d1.equals("1-1-2024") must false

```
class Date {
    private int d, m, y;
    // constructors, getters/setters

    public boolean equals(Object other) {
        if (other == null)
            return false; // rule 5
        if (this == other)
            return true; // rule 1
        if (!(other instanceof Date))
            return false; // rule 6
        Date that = (Date) other;
        if (this.d == that.d && this.m == that.m &&
                this.y == that.y) // state comparison
            return true;
        return false;
    }
}
```

p
Person
Upcasting
Employee
emp
"Sandeep"
String
39
int
378
int
45000.50
float
name
age
empid
salary
[Employee Instance]
Instance of Person class which is
Exist inside Employee class.

The instance, in which space is reserved for name & age and we can call only showRecord method on it is an instance of Person class.

p
Pereson
Downcasting
Employee
emp
"Sandeep"
String
39
int
378
int
45000.50
float
name
age
empid
salary
[Employee Instance]
Instance of Person class which is
Exist inside Employee class.

[Shape class instance which is
Exist inside Rectangle instance]
rect
Rectangle
200.0
float
area
Downcasting
10.0
float
length
20.0
float
breadth
shape
null
Shape
[Rectangle Instance]
[Shape class instance which is
Exist inside Circle instance]
Downcasting
314.0
float
area
circle
Circle
10.0
float
radius
[Circle Instance]

Service Providers
Surya
Orpat
Phillips
Bajaj
Syska
Seema
interface
ISI
Service Consumer
Sandeep
Document
Set of rules
/ Specification
/Standard
1. Power Consumption
2. Warranty
3. Price

Service Providers
Apple C Compiler(cc)
Serive Consumer
interface
Linux Compiler(gcc)
C Program
ANSI
Turbo C Compiler
Microsoft C Compiler(cl.exe)

Service Provider
MS SQL Server
Service Consumer
interface
Oracle
Java Application
JDBC
MySQL
Other RDBMS

Inheritance

To avoid ambiguity errors due to multiple inheritance, Java lanaguage doesn't have support for multiple implementation(class) inheritance. However, Java allows multiple interface inheriance.

All members of parent class are inherited to the child class.

Parent/Super class : generalized

inheritance

Child/Sub class : Specialized

super keyword:

1. invoke super class ctor from sub class ctor.
2. access super class members from sub class non-static methods.

```
class Person {
    =
    public void display() {
        =
    }
}
class Student extends Person {
    =
    public void display() {
        super.display();
        =
    }
}
main():
    Student s1 = new Student();
    s1.display();
```

// cannot call Person.display on Student ref → Method shadowing

In C++:

```
class Programmer {
    public:
        void display() { ... }
};
class Teacher {
    public:
        void display() { ... }
};
class ProgrammingTeacher:
    public Programmer, Teacher {
    // ...
};
main():
ProgrammingTeacher obj;
obj.display(); // ambiguity error
```

File Edit Selection View Go ··· private

day07.md × day06.md

day07.md > # Core Java > ## Polymorphism > ### Method overriding

```
        /*protected or*/ public Number calculate(Integer i, Float f) {
            // ...
        }
    }
```

* Arguments of sub-class method must be same as of super-class method. The return-type of sub-class method can be same or sub-class of the super-class's method's return-type. This is called as "covarient" return-type.

```Java
    class SuperClass {
        public Number calculate(Integer i, Float f) {
            // ...
        }
    }
    class SubClass extends SuperClass {
        // Double is inherited from Numer (i.e. return-type of super-class method)
        public Double calculate(Integer i, Float f) {
            // ...
        }
    }
```

Object
↑
Number
↑
Double

* Checked exception list in sub-class method should be same or subset of exception list in super-class method.

```Java
    class SuperClass {
        public void testMethod() throws IOException, SQLException {
```

main* ⊗ 0 ⚠ 0 0 Ln 167, Col 23 Spaces: 4 UTF-8 CRLF Markdown

Search 11:46 AM

day08 - demo03/src/com/sunbeam/Rectangle.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

interfaces -- to define standards/specifications/rules.

Program03.java | Shape.java

```java
package com.sunbeam;

public interface Shape {
    /*public abstract*/ double calcArea();
    /*public abstract*/ double calcPeri();
}
```

interface contains only method declarations.

interface methods must be implemented in sub-classes. Otherwise, sub-class will be abstract class.

interfaces are immutable i.e. once published interface should not be modified.

if you need additional functionality in an interface, create a new interface inherited from the interface and add fn there.

Rectangle.java

interface inheritance

```java
package com.sunbeam;

public class Rectangle implements Shape {
    private double length;
    private double breadth;
    public Rectangle() {
        this.length = 0;
        this.breadth = 0;
    }
    public Rectangle(double length, double brea
        this.length = length;
        this.breadth = breadth;
    }
    @Override
    public double calcArea() {
        return this.length * this.breadth;
    }
    @Override
    public double calcPeri() {
        return 2 * (this.length + this.breadth
    }
    public double getLength() {
        return length;
```

Writable | Smart Insert | 22 : 1 : 466

Search 11:35 AM

Core Java

equals() method

- Non-final method of java.lang.Object class.
  - public boolean equals(Object other);
- Definition of Object.equals():

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- To compare the object contents/state, programmer should override equals() method.
- This equals() must have following properties:
  - Reflexive: for any non-null reference value x, x.equals(x) should return true.
  - Symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
  - Transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
  - Consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
  - For any non-null reference value x, x.equals(null) should return false.
- Example:

```
class Employee {
    // ...
    @Override
    public boolean equals(Object obj) {
        if(obj == null)
            return false;

        if(this == obj)
            return true;

        if(! (obj instanceof Employee))
            return false;

        Employee other = (Employee) obj;
        if(this.id == other.id)
            return true;
        return false;
    }
}
```

Garbage collection

- Garbage collection is automatic memory management by JVM.
- If a Java object is unreachable (i.e. not accessible through any reference), then it is automatically released by the garbage collector.
- An object become eligible for GC in one of the following cases:
  - Nullify the reference.

    ```
    MyClass obj = new MyClass();
    obj = null;
    ```

  - Reassign the reference.

    ```
    MyClass obj = new MyClass();
    obj = new MyClass();
    ```

  - Object created locally in method.

    ```
    void method() {
        MyClass obj = new MyClass();
        // ...
    }
    ```

- GC is a background thread in JVM that runs periodically and reclaim memory of unreferenced objects.
- Before object is destroyed, its finalize() method is invoked (if present).
- One should override this method if object holds any resource to be released explicitly e.g. file close, database connection, etc.

  ```
  class MyClass {
      private Connection con;
      public MyClass() throws Exception {
          con = DriverManager.getConnection("url", "username", "password");
      }
      // ...
      @Override
      public void finalize() {
          try {
              if(con != null)
                  con.close();
          }
          catch(Exception e) {
  ```

```
        }
    }
}
class Main {
    public static void method() throws Exception {
        MyClass my = new MyClass();
        my = null;
        System.gc(); // request GC
    }
    // ...
}
```

- GC can be requested (not forced) by one of the following.
    - System.gc();
    - Runtime.getRuntime().gc();
- JVM GC internally use Mark and Compact algorithm.
- GC Internals: https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Marker interfaces

- Interface that doesn't contain any method declaration is called as "Marker interface".
- These interfaces are used to mark or tag certain functionalities/features in implemented class. In other words, they associate some information (metadata) with the class.
- Marker interfaces are used to check if a feature is enabled/allowed for the class.
- Java has a few pre-defined marker interfaces. e.g. Serializable, Cloneable, etc.
    - java.io.Serializable -- Allows JVM to convert object state into sequence of bytes.
    - java.lang.Cloneable -- Allows JVM to create copy of the class object.

Cloneable interface

- Enable creating copy/clone of the object.
- If a class is Cloneable, Object.clone() method creates a shallow copy of the object. If class is not Cloneable, Object.clone() throws CloneNotSupportedException.
- A class should implement Cloneable and override clone() to create a deep/shallow copy of the object.

```
class Date implements Cloneable {
    private int day, month, year;
    // ...
    // shallow copy
    public Object clone() throws CloneNotSupportedException {
        Date temp = (Date)super.clone();
        return temp;
    }
}
```

```
class Person implements Cloneable {
    private String name;
    private int weight;
    private Date birth;
    // ...
    // deep copy
    public Object clone() throws CloneNotSupportedException {
        Person temp = (Person)super.clone(); // shallow copy
        temp.birth = (Date)this.birth.clone(); // + copy reference types
explicitly
        return temp;
    }
}
```

```
class Program {
    public static void main(String[] args) throws CloneNotSupportedException
{
        Date d1 = new Date(28, 9, 1983);
        System.out.println("d1 = " + d1.toString());
        Date d2 = (Date)d1.clone();
        System.out.println("d2 = " + d2.toString());
        Person p1 = new Person("Nilesh", 70, d1);
        System.out.println("p1 = " + p1.toString());
        Person p2 = (Person)p1.clone();
        System.out.println("p2 = " + p2.toString());
    }
}
```

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
    }
}
```

equals() overriding

Rules for overriding equals()

① equals() must be reflexive.
d1.equals(d1) must be true.

② equals() must be symmetric.
if d1.equals(d2) is true,
then d2.equals(d1) must be true.

③ equals() must be transitive.
if d1.equals(d2) is true and
d2.equals(d3) is true, then
d1.equals(d3) must be true.

④ equals() must be consistent
Result of d1.equals(d2) should
remain same until any
of the object state modified.

⑤ null comparison
d1.equals(null) must be false.

⑥ Comparison with incompatible type.
d1.equals("1-1-2024") must false

```
class Date {
    private int d, m, y;
    // constructors, getters/setters

    public boolean equals(Object other) {
        if (other == null)
            return false; // rule 5
        if (this == other)
            return true; // rule 1
        if (!(other instanceof Date))
            return false; // rule 6
        Date that = (Date) other;
        if (this.d == that.d && this.m == that.m &&
                this.y == that.y) // state comparison
            return true;
        return false;
    }
}
```

[main method]
[equals method]
this
Employee
[Heap Section]
emp1
Employee
"Sandeep"
String
name
378
int
empid
65000.50
float
salary
[Upcasting]
obj
Object
emp2
Employee
[Downcasting]
Employee
other
"Sandeep"
String
name
378
int
empid
65000.50
float
salary

day08 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

In this example, since Date is not inherited from Cloneable, its copy will not be created and will throw ex.

Date.java

```java
package com.sunbeam;

public class Date extends Object {
    private int day, month, year;
    public Date() {
        this(1, 1, 2000);
    }
    public Date(int day, int month, int year) {
        this.day = day;
        this.month = month;
        this.year = year;
    }
    @Override
    public Object clone() throws CloneNotSuppo
        Object temp = super.clone(); //Object.
        return temp;
    }
    public int getDay() {
        return day;
    }
    public void setDay(int day) {
        this.day = day;
    }
```

Program04.java

```java
com.sunbeam;

lass Program04 {
ic static void main(String[] args) throws Clon
Date d1 = new Date(1, 2, 2024);
Date d2 = (Date) d1.clone();
System.out.println("d1: " + d1.toString());
System.out.println("d2: " + d2.toString());


// pre-defined Object class
class Object {
    // ...
    Object clone() throws ... {
        if(! (this instanceof Cloneable))
            throw CloneNotSupportedException;
        // create copy of "this" object and return
    }
}
```

Writable | Smart Insert | 8 : 39 : 273

Search 12:54 PM

day08 - demo04/src/com/sunbeam/Date.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Date.java

```java
package com.sunbeam;

public class Date implements Cloneable {
    private int day, month, year;
    public Date() {
        this(1, 1, 2000);
    }
    public Date(int day, int month, int year) {
        this.day = day;
        this.month = month;
        this.year = year;
    }
    @Override
    public Object clone() throws CloneNotSuppor
        Object temp = super.clone(); //Object.c
        return temp;
    }
    public int getDay() {
```

? - checked

shallow copy...

Program04.java

```java
e com.sunbeam;

 class Program04 {
blic static void main(String[] args) throws Clo
  Date d1 = new Date(1, 2, 2024);
  Date d2 = (Date) d1.clone();
  System.out.println("d1: " + d1.toString());
  System.out.println("d2: " + d2.toString());
```

destroyed when Date.clone() returns

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 7, 2024, 12:58:56 PM – 12:58:56 PM) [pid: 12816]

```
d1: 1-2-2024
d2: 1-2-2024
```

Writable | Smart Insert | 3 : 39 [20]

Search 1:03 PM

day09 - demo01/src/com/sunbeam/Person.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Deep copy - Copy of outer object as well as inner objects (referred by the fields in outer object). Now both objects have different memory locations. Changes in one will not affect other object.

Date.java | Person.java

```java
3 public class Person implements Cloneable {
4     private String name;
5     private int height, weight;
6     private Date birth;
7
8     public Person() {
11    public Person(String name, int height, int wei
17    public int getHeight() {
20    public void setHeight(int height) {
23    public int getWeight() {
26    public void setWeight(int weight) {
29    public String getName() {
32    public void setName(String name) {
35    public Date getBirth() {
38    public void setBirth(Date birth) {
42    public String toString() {
45
46    @Override
47    protected Object clone() throws CloneNotSuppor
48        Person temp = (Person) super.clone();
49        temp.birth = (Date) this.birth.clone();
50        return temp;
51    }
```

Program01.java

```java
1 package com.sunbeam;
2
3 public class Program01 {
4     public static void main(String[] args) throws CloneNotSupportedException {
5         Person p1 = new Person("Nilesh", 175, 75, new Date(28, 9, 1983));
6         Person p2 = (Person) p1.clone();
7         System.out.println("p1: " + p1.toString());
8         System.out.println("p2: " + p2.toString());
9         p2.getBirth().setDay(1);
10        p2.getBirth().setMonth(2);
11        System.out.println("p1: " + p1.toString());
12        System.out.println("p2: " + p2.toString());
13    }
14 }
15
```

Problems | Javadoc | Declaration | Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 8, 2024, 9:15:17 AM – 9:15:17 AM) [pid: 2940]

```
p1: Person [name=Nilesh, height=175, weight=75, birth=28-9-1983]
p2: Person [name=Nilesh, height=175, weight=75, birth=28-9-1983]
p1: Person [name=Nilesh, height=175, weight=75, birth=1-2-1983]
p2: Person [name=Nilesh, height=175, weight=75, birth=1-2-1983]
```

Writable | Smart Insert | 45 : 5 : 945

Search | 9:26 AM

day09 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - src
    - com.sunbeam
      - Date.java
      - Person.java
      - Program01.java
- demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - src
    - com.sunbeam
      - Program02.java

Program02.java

```java
package com.sunbeam;

public class Program02 {
    public static void main(String[] args) {
        // String class in Java -- represents immutable "sequence of characters"
        // length() returns number of chars
        // charAt() returns char at given index -- 0 to length()-1
        // "str" reference is created on "stack"
        // "Sunbeam" string literal/constant is created on String pool (in heap)
        String str = "Sunbeam";
        System.out.println("Length: " + str.length());
        for(int i=0; i<str.length(); i++) {
            char ch = str.charAt(i);
            System.out.print(ch);
        }
    }
}
```

String st = new String("Infotech");

The "new" string objects are created on Heap.

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 8, 2024, 9:47:2

```
Length: 7
Sunbeam
```

Writable | Smart Insert | 9 : 81 : 388

Search 9:51 AM

String objects vs String literals

* Example 01:

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

* Example 02:

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

* Example 03:

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

* Example 04:

```
```
```

* Example 04:

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

evaluated by compiler
"Sunbeam" is already in pool.

* Example 05:

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

compiler generate byte code
evaluated by jvm at runtime

new obj in heap

* Example 06:

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

* Example 07:
```

File Edit Selection View Go ··· private

day09.md × day08.md classwork.md U Untitled-1 ●

day09.md > # Core Java > ## Java Strings > ### String objects vs String literals

```Java
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));
```

In Java string contents are case sensitive.

two different objects created in String pool.

// ??? false

boolean

// ??? false ← case sensitive comparison

// ??? true ← case insensitive comparison

int (differnce)

// ??? returns difference of first non-matching char i.e. 'b' - 'B' = 32

// ???returns difference of first non-matching char but case-insensitive i.e. 0

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)

main* ⊗ 0 ⚠ 0 0 Ln 85, Col 22 Spaces: 4 UTF-8 CRLF Markdown

Search 10:50 AM

File Edit Selection View Go ··· private

day09.md × day08.md classwork.md U Untitled-1 ●

day09.md > # Core Java > ## Java Strings > ### String objects vs String literals

```Java
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));
```

In Java string contents are case sensitive.

two different objects created in String pool.

// ??? false

boolean

// ??? false ← case sensitive comparison

// ??? true ← case insensitive comparison

int (differnce)

// ??? returns difference of first non-matching char i.e. 'b' - 'B' = 32

// ???returns difference of first non-matching char but case-insensitive i.e. 0

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)

main* ⊗ 0 ⚠ 0 0 Ln 85, Col 22 Spaces: 4 UTF-8 CRLF Markdown

Search 10:50 AM

day09 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - src
    - com.sunbeam
      - Date.java
      - Person.java
      - Program01.java
- demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - src
    - com.sunbeam
      - Program02.java
- demo03 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - src
    - com.sunbeam
      - Program03.java

Program02.java | Program03.java

```java
package com.sunbeam;

public class Program03 {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();
        sb.append("Nilesh"); // append(String)
        sb.append(40); // append(int)
        sb.append('M'); // append(char)
        sb.append(75.45); // append(double)
        String str = sb.toString();
        System.out.println(str);
    }

    /*
    public static void main(String[] args) {
        // capacity is size of internal char array
        // length is number of chars stored in that array
        StringBuffer sb = new StringBuffer();
        System.out.println("Capacity: " + sb.capacity() + ", Length: " + sb.length()); // Capacity: 2
```

Problems | Javadoc | Declaration | Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 8, 2024, 11:23

```
Nilesh40M75.45
```

Writable | Smart Insert | 11 : 31 : 338

Search | 11:23 AM

day09 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java | Program03.java

```
        System.out.println(str);
    }
    */

    public static void main(String[] args) {
        // capacity is size of internal char array
        // length is number of chars stored in that array
        StringBuffer sb = new StringBuffer();
        System.out.println("Capacity: " + sb.capacity() + ", Length: " + sb.length()); // Capacity: 16, Length: 0
        sb.append("0123456789");
        System.out.println("Capacity: " + sb.capacity() + ", Length: " + sb.length()); // Capacity: 16, Length: 10
        sb.append("ABCDEF");
        System.out.println("Capacity: " + sb.capacity() + ", Length: " + sb.length()); // Capacity: 16, Length: 16
        sb.append("GHIJKL");
        System.out.println("Capacity: " + sb.capacity() + ", Length: " + sb.length()); // Capacity: 34, Length: 22
    }

    /*
    public static void main(String[] args) {
```

buffer capacity is full

when appended more data, buffer will be expanded/grow.

new capacity = (current capacity + 1) * 2 = (16 + 1) * 2 = 34

Problems | Javadoc | Declaration | Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 8, 2024, 11:31:09 AM – 11:31:09 AM) [pid: 2972]

```
Capacity: 16, Length: 10
Capacity: 16, Length: 16
Capacity: 34, Length: 22
```

Writable | Smart Insert | 24 : 115 : 921

Search | 11:31 AM

day09 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java | Program03.java

```java
//                  return sb.toString();
//              }
            public String toString() {
                // more efficient than StringBuffer -- Since Java 5.0
                StringBuilder sb = new StringBuilder();
                String str = sb.append("Box: length=")
                    .append(this.length)
                    .append(", breadth=")
                    .append(this.breadth)
                    .append(", height=")
                    .append(this.height)
                    .toString();
                return str;
            }
        }
        Box b = new Box(5, 4, 3);
        System.out.println("b = " + b.toString());
    }
}
```

StringBuffer/StringBuilder's most of the methods return the current object itself.

It is possible to invoke methods in a chain/cascaded-style.

Here "str" String is created using StringBuilder object's various operations. In other words, we gave instructions to StringBuilder about creating the String and it created String object accordingly (toString() method).

This is called as - Builder Design Pattern.

Problems | Javadoc | Declaration | Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 8, 2024, 11:46:16 AM – 11:46:16 AM) [pid: 4216]

```
sb1 == sb2 : false
sb1.equals(sb2) : false
```

Writable | Smart Insert | 99 : 28 : 3727

Search | 11:55 AM

* Example 04:

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

reference comparison

Object.equals() -- reference comparison

* Example 05:

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

* Example 06:

```Java
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
```

```
    }
    ```

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.
    ```Java
    class MyResource implements AutoCloseable {
        // ...
        public void close() {
```

```
public class Program04 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.o                                    );
        int num
    }

}
```

Resource leak: 'sc' is never closed

4 quick fixes available:

- Surround with try-with-resources
- Add @SuppressWarnings 'resource' to 'sc'
- Add @SuppressWarnings 'resource' to 'main()'
- Configure problem severity

day09 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - src
    - com.sunbeam
      - Date.java
      - Person.java
      - Program01.java
- demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - src
    - com.sunbeam
      - Program02.java
- demo03 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - src
    - com.sunbeam
      - Program03.java
- demo04 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - src
    - com.sunbeam
      - Program04.java

Program04.java

```
        // "sc" is not closed -- stdin in not closed -- resource leakage
    }
    */

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Entet a number: ");
        int num = sc.nextInt();
        System.out.println("Square: " + num * num);
        sc.close(); // internally close System.in i.e. stdin
    }

}
```

if user give wrong input, program will be aborted with exception and sc.close() is not called.
in this case resource will leak.

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 8, 2024, 12:26

```
Entet a number: three
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at com.sunbeam.Program04.main(Program04.java:19)
```

Writable | Smart Insert | 24 : 2 : 624

Search 12:26 PM

day09 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- > demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Date.java
      - Person.java
      - Program01.java
- > demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program02.java
- > demo03 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program03.java
- > demo04 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program04.java

Program04.java

```java
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Entet a number: ");
            int num = sc.nextInt();
            System.out.println("Square: " + num * num);
            System.exit(0);
        }
        finally {
            System.out.println("Closing scanner.");
            sc.close(); // internally close System.in i.e. stdin
        }
    }
```

if application JVM exits, then finally block is not executed. Rest all cases (like return from method, break, ...) finally will execute.

finally block is always executed irrespective of exception occurrs or not.

Problems | Javadoc | Declaration | Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 8, 2024, 12:30

```
Entet a number: three
Closing scanner.
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at com.sunbeam.Program04.main(Program04.java:30)
```

Writable | Smart Insert | 32 : 13 : 900

Search 12:30 PM

day09 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main
- demo02 [CJ-H-02 main
- demo03 [CJ-H-02 main
- demo04 [CJ-H-02 main
  - JRE System Library [Jav
  - src
    - com.sunbeam
      - Program04.java

Program04.java

```java
            sc.close(); // internally close System.in i.e. stdin
        }
    }
    */
    public static void main(String[] args) {
        // try-with-resource (since Java 7) ensure that resource is auto-closed.
        try(Scanner sc = new Scanner(System.in)) {
            System.out.print("Entet a number: ");
            int num = sc.nextInt();
            System.out.println("Square: " + num * num);
        } // sc.close(); // called automatically
    }
}
```

try-with-resource works with any class that is inherited from AutoCloseable.

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 8, 2024, 12:38:49 PM

```
Entet a number: 4
Square: 16
```

Search 12:39 PM

day09 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main
- demo02 [CJ-H-02 main
- demo03 [CJ-H-02 main
- demo04 [CJ-H-02 main
  - JRE System Library [Jav
  - src
    - com.sunbeam
      - Program04.java

Program04.java

```java

    public static void main(String[] args) {
        // custom resource class -- auto-closeable
        class MyResource implements AutoCloseable {
            // ...
            public MyResource() {
                System.out.println("MyResource created.");
            }
            public void close() {
                System.out.println("MyResource closed.");
            }
        }

        try(MyResource mr = new MyResource()) {
            System.out.println("using MyResource");
        } // mr.close();
    }
}
```

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 8, 2024, 12:51:25 PM

```
MyResource created.
using MyResource
MyResource closed.
```

Writable | Smart Insert | 53 : 51 : 1539

Search 12:52 PM

Organize New Open Select

esh > onlinecourses > CJ-H-02 > day09 > demo04 > bin > com > sunbeam

Search sunbea

Name

Program04$1MyResource.class

Program04.class

class MyResource ... { ... }
-- local class.

class Program04 { ... }

There is one .class file created for every class.
Top-level class as well as Local class.

m

automatically released by the garbage collector.

* An object become eligible for GC in one of the following cases:
    * Nullify the reference.

```Java
MyClass obj = new MyClass();
obj = null;
```

    * Reassign the reference.

```Java
MyClass obj = new MyClass();
obj = new MyClass();
```

    * Object created locally in method.

```Java
void method() {
    MyClass obj = new MyClass();
    // ...
} ⟵ returns
```

    * Island of isolation i.e. objects are referencing each other, but not referenced

day10 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java

```java
class MyClass {
    // the resource
    public MyClass() {
        System.out.println("Resource allocated.");
    }
    // ...
    // When GC destroys object, it will call finalize() before that.
    @Override
    protected void finalize() throws Throwable {
        // close the resource
        System.out.println("Resource closed.");
    }
}
public class Program01 {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj = null; // MyClass obj created above is eligible for GC
        System.gc(); // requests GC.
        System.out.println("Bye!");
    }
```

(1) execute finalize()

Problems Javadoc Declaration Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 9, 2024, 9:37:36 AM – 9:37:36 AM) [pid: 15632]

```
Resource allocated.
Bye!
Resource closed.
```

Search 9:38 AM

Core Java

Garbage collection

- Garbage collection is automatic memory management by JVM.
- If a Java object is unreachable (i.e. not accessible through any reference), then it is automatically released by the garbage collector.
- An object become eligible for GC in one of the following cases:
  - Nullify the reference.

    ```
    MyClass obj = new MyClass();
    obj = null;
    ```

  - Reassign the reference.

    ```
    MyClass obj = new MyClass();
    obj = new MyClass();
    ```

  - Object created locally in method.

    ```
    void method() {
        MyClass obj = new MyClass();
        // ...
    }
    ```

- GC is a background thread in JVM that runs periodically and reclaim memory of unreferenced objects.
- Before object is destroyed, its finalize() method is invoked (if present).
- One should override this method if object holds any resource to be released explicitly e.g. file close, database connection, etc.

  ```
  class MyClass {
      private Connection con;
      public MyClass() throws Exception {
          con = DriverManager.getConnection("url", "username", "password");
      }
      // ...
      @Override
      public void finalize() {
          try {
  ```

```
            if(con != null)
                con.close();
        }
        catch(Exception e) {
        }
    }
}
class Main {
    public static void method() throws Exception {
        MyClass my = new MyClass();
        my = null;
        System.gc(); // request GC
    }
    // ...
}
```

- GC can be requested (not forced) by one of the following.
    - System.gc();
    - Runtime.getRuntime().gc();
- JVM GC internally use Mark and Compact algorithm.
- GC Internals: https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

JVM Architecture

Java compilation process

- Hello.java --> Java Compiler --> Hello.class
    - javac Hello.java
- Java compiler converts Java code into the Byte code.

Byte code

- Byte code is machine level instructions that can be executed by Java Virtual Machine (JVM).
    - Instruction = Op code + Operands
        - e.g. iadd op1, op2
- Each Instruction in byte-code is of 1 byte.
    - .class --> JVM --> Windows (x86)
    - .class --> JVM --> Linux (ARM)
- JVM converts byte-code into target machine/native code (as per architecture).

.class format

- .class file contains header, byte-code, meta-data, constant-pool, etc.
- .class header contains
    - magic number -- 0xCAFEBABE (first 4 bytes of .class file)
    - information of other sections
- .class file can inspected using "javap" tool.

    - terminal> javap java.lang.Object
        - Shows public and protected members
    - terminal> javap -p java.lang.Object
        - Shows private members as well
    - terminal> javap -c java.lang.Object
        - Shows byte-code of all methods
    - terminal> javap -v java.lang.Object
        - Detailed (verbose) information in .class
            - Constant pool
            - Methods & their byte-code
            - ...
- "javap" tool is part of JDK.

Executing Java program (.class)

- terminal> java Hello
- "java" is a Java Application Launcher.
- java.exe (disk) --> Loader --> (Windows OS) Process
- When "java" process executes, JVM (jvm.dll) gets loaded in the process.
- JVM will now find (in CLASSPATH) and execute the .class.

JVM Architecture (Overview)

- JVM = Classloader + Memory Areas + Execution Engine

Classloader sub-system

- Load and initialize the class

Loading

- Three types of classloaders
    - Bootstrap classloader: Load Java builtin classes from jre/lib jars (e.g. rt.jar).
    - Extended classloader: Load extended classes from jre/lib/ext directory.
    - Application classloader: Load classes from the application classpath.
- Reads the class from the disk and loads into JVM method (memory) area.

Linking

- Three steps: Verification, Preparation, Resolution
- Verification: Byte code verifier does verification process. Ensure that class is compiled by a valid compiler and not tampered.
- Preparation: Memory is allocated for static members and initialized with their default values.
- Resolution: Symbolic references in constant pool are replaced by the direct references.

Initialization

- Static variables of the class are assigned with given values (field initializers).

- Execute static blocks if present.

JVM memory areas

- During execution, memory is required for byte code, objects, variables, etc.
- There are five areas: Method area, Heap area, Stack area, PC Registers, Native Method Stack area.

Method area

- Created during JVM startup.
- Shared by all threads (global).
- Class contents (for all classes) are loaded into Method area.
- Method area also holds constant pool for all loaded classes.

Heap area

- Created during JVM startup.
- Shared by all threads (global).
- All allocated objects (with new keyword) are stored in heap.
- The class Metadata is stored in a java.lang.Class object (in heap) once class is loaded.
- The string pool is part of Heap area.

Stack area

- Separate stack is created for each thread in JVM (when thread is created).
- When a method is called from the stack, a new FAR (stack frame) is created on its stack.
- Each stack frame conatins local variable array, operand stack, and other frame data.
- When method returns, the stack frame is destroyed.

PC Registers

- Separate PC register is created for each thread. It maintains address of the next instruction executed by the thread.
- After an instruction is completed, the address in PC is auto-incremented.

Native method stack area

- Separate native method stack is created for each thread in JVM (when thread is created).
- When a native method is called from the stack, a stack frame is created on its stack.

Execution engine

- The main comonent of JVM.
- Execution engine executes for executing Java classes.

Interpreter

- Convert byte code into machine code and execute it (instruction by instruction).

- Each method is interpreted by the interpreter at least once.
- If method is called frequently, interpreting it each time slow down the execution of the program.
- This limitation is overcomed by JIT (added in Java 1.1).

JIT compiler

- JIT stands for Just In Time compiler.
- Primary purpose of the JIT compiler to improve the performance.
- If a method is getting invoked multile times, the JIT compiler convert it into native code and cache it.
- If the method is called next time, its cached native code is used to speedup execution process.

Profiler

- Tracks resource (memory, threads, ...) utilization for execution.
- Part of JIT that identifies hotspots. It counts number of times any method is executing. If the number is more than a threshold value, it is considered as hotspot.

Garbage collector

- When any object is unreferenced, the GC release its memory.

JNI

- JNI acts as a bridge between Java method calls and native method implementations.

Exception Handling

- Exceptions represents runtime problems.
- If these problems may not be handled in the current method, so they can be sent back to the calling method.
- Java keywords for exception handling
  - throw
  - try
  - catch
  - finally
  - throws
- Example 1:

```
static double divide(int numerator, int denominator) {
    if(denominator == 0)
        throw new RuntimeException("Cannot divide by zero");
    return (double)numerator / denominator;
}
public static void main(String[] args) {
    // ...
    try {
        int num1 = sc.nextInt();
        int num2 = sc.nextInt();
```

```
        double result = divide(num1, num2);
        System.out.println("Result: " + result);
    }
    catch(RuntimeException e) {
        e.printStackTrace();
    }
}
```

- Java operators/APIs throw pre-defined exception if runtime problem occurs.
- For example, ArithmeticException is thrown when divide by zero is tried.
- Example 2:

```
static int divide(int numerator, int denominator) {
    return numerator / denominator;
}
public static void main(String[] args) {
    // ...
    try {
        int num1 = sc.nextInt();
        int num2 = sc.nextInt();
        int result = divide(num1, num2);
        System.out.println("Result: " + result);
    }
    catch(ArithmeticException e) {
        e.printStackTrace();
    }
}
```

- Java exception class hierarchy

```
Object
    |- Throwable
        |- Error
        |   |- AssertionError
        |   |- VirtualMachineError
        |           |- StackOverflowError
        |           |- OutOfMemoryError
        |- Exception
            |- CloneNotSupportedException
            |- IOException
            |       |- EOFException
            |       |- FileNotFoundException
            |- SQLException
            |- InterruptedException
            |- RuntimeException
                |- NullPointerException
                |- ArithmeticException
                |- NoSuchElementException
                |       |- InputMismatchException
```

```
                    |- IndexOutOfBoundsException
                            |- ArrayIndexOutOfBoundsException
                            |- StringIndexOutOfBoundsException
```

- One catch block cannot handle problems from multiple try blocks.
- One try block may have multiple catch blocks. Specialized catch block must be written before generic catch block.
- If certain code to be executed irrespective of exception occur or not, write it in finally block.
- Example:

```java
try {
    // file read code -- possible problems
        // 1. file not found
        // 2. end of file is reached
        // 3. error while reading from file
        // 4. null reference (programmer's mistake)
}
catch(NullPointerException ex) {
    // ...
}
catch(FileNotFoundException ex) {
    // ...
}
catch(EOFException ex) {
    // ...
}
catch(IOException ex) {
    // ...
}
finally {
    // close the file
}
```

- When exception is raised, it will be caught by nearest matching catch block. If no matching catch block is found, the exception will be caught by JVM and it will abort the program.

java.lang.Throwable class

- Throwable is root class for all errors and exceptions in Java.
- Only objects of
- Methods
    - Throwable()
    - Throwable(String message)
    - Throwable(Throwable cause)
    - Throwable(String message, Throwable cause)
    - String getMessage()
    - void printStackTrace()
    - void printStackTrace(PrintStream s)

    - void printStackTrace(PrintWriter s)
    - String toString()
    - Throwable getCause()

java.lang.Error class

- Usually represents the runtime problems that are not recoverable.
- Generated due to environmental condition/Runtime environment (e.g. OS error, Memory error, etc.)
- Examples:
    - AssertionError
    - VirtualMachineError
    - StackOverflowError
    - OutOfMemoryError

java.lang.Exception class

- Represents the runtime problems that can be handled.
- The exception is handled using try-catch block.
- Examples:
    - CloneNotSupportedException
    - IOException
    - SQLException
    - NullPointerException
    - ArrayIndexOutOfBoundsException
    - ClassCastException

Exception types

- There are two types of exceptions
    - Checked exception -- Checked by compiler and forced to handle
    - Unchecked exception -- Not checked by compiler

Unchecked exception

- RuntimeException and all its sub classes are unchecked exceptions.
- Typically represents programmer's or user's mistake.
    - NullPointerException, NumberFormatException, ClassCastException, etc.
- Compiler doesn't provide any checks -- if exception is handled or not.
- Programmer may or may not handle (catch block) the exception. If exception is not handled, it will be caught by JVM and abort the application.

Checked exception

- Exception and all its sub classes (except RuntimeException) are checked exceptions.
- Typically represents problems arised out of JVM/Java i.e. at OS/System level.
    - IOException, SQLException, InterruptedException, etc.
- Compiler checks if the exception is handled in one of following ways.
    - Matching catch block to handle the exception.

```
void someMethod() {
    try {
        // file io code
    }
    catch(IOException ex) {
        // ...
    }
}
```

- throws clause indicating exception to be handled by calling method.

```
void someMethod() throws IOException {
    // file io code
}
void callingMethod() {
    try {
        someMethod();
    }
    catch(IOException ex) {
        // ...
    }
}
```

Exception handling keywords

- "try" block
  - Code where runtime problems may arise should be written in try block.
  - try block must have one of the following
    - catch block
    - finally block
    - try-with-resource
  - Can be nested in try, catch, or finally block.
- "throw" statement
  - Throws an exception/error i.e. any object that is inherited from Throwable class.
  - Can throw only one exception at time.
  - All next statements are skipped and control jumps to matching catch block.
- "catch" block
  - Code to handle error/exception should be written in catch block.
  - Argument of catch block must be Throwable or its sub-class.
  - Generic catch block -- Performs upcasting -- Should be last (if multiple catch blocks)

```
try {
    // ...
}
catch(Throwable e) {
```

```
    // can handle exception of any type
}
```

- Multi-catch block -- Same logic to execute different exceptions

```
try {
    // ...
}
catch(ArithmeticException|InputMismatchException e) {
    // can handle exception of any type
}
```

- Exception sub-class should be caught before super-class.

```
try {
    // ...
}
catch(EOFException ex) {
    // ...
}
catch(IOException ex) {
    // ...
}
```

- "finally" block
  - Resources are closed in finally block.
  - Executed irrespective of exception occurred or not.
  - finally block not executed if JVM exits (System.exit()).

```
Scanner sc = new Scanner(System.in);
try {
    // ...
}
catch(Exception ex) {
    // ...
}
finally {
    sc.close();
}
```

- "throws" clause
  - Written after method declaration to specify list of exception not handled by called method and to be handled by calling method.
  - Writing unhandled checked exceptions in throws clause is compulsory. The unchecked exceptions written in throws clause are ignored by the compiler.

```
void someMethod() throws IOException, SQLException {
    // ...
}
```

- Sub-class overridden method can throw same or subset of exception from super-class method.

```
class SuperClass {
    // ...
    void method() throws IOException, SQLException,
InterruptedException {

    }
}
class SubClass extends SuperClass {
    // ...
    void method() throws IOException, SQLException {

    }
}
```

```
class SuperClass {
    // ...
    void method() throws IOException {

    }
}
class SubClass extends SuperClass {
    // ...
    void method() throws FileNotFoundException, EOFException {

    }
}
```

Exception Handling

Exception chaining

- Sometimes an exception is generated due to another exception.
- For example, database SQLException may be caused due to network problem SocketException.
- To represent this an exception can be chained/nested into another exception.
- If method's throws clause doesn't allow throwing exception of certain type, it can be nested into another (allowed) type and thrown.

User defined exception class

- If pre-defined exception class are not suitable to represent application specific problem, then user-defined exception class should be created.
- User defined exception class may contain fields to store additional information about problem and methods to operate on them.
- Typically exception class's constructor call super class constructor to set fields like message and cause.
- If class is inherited from RuntimeException, it is used as unchecked exception. If it is inherited from Exception, it is used as checked exception.

Generic Programming

- Code is said to be generic if same code can be used for various (practically all) types.
- Best example:
  - Data structure e.g. Stack, Queue, Linked List, ...
  - Algorithms e.g. Sorting, Searching, ...
- Two ways to do Generic Programming in Java
  - using java.lang.Object class -- Non typesafe
  - using Generics -- Typesafe

Generic Programming Using java.lang.Object

```
```Java
class Box {
    private Object obj;
    public void set(Object obj) {
        this.obj = obj;
    }
    public Object get() {
        return this.obj;
    }
}
```
```Java
Box b1 = new Box();
b1.set("Nilesh");
String obj1 = (String)b1.get();
System.out.println("obj1 : " + obj1);

Box b2 = new Box();
b2.set(new Date());
Date obj2 = (Date)b2.get();
System.out.println("obj2 : " + obj2);

Box b3 = new Box();
b3.set(new Integer(11));
String obj3 = (String)b3.get();  // ClassCastException
System.out.println("obj3 : " + obj3);
```
```

Generic Programming Using Generics

- Added in Java 5.0.
- Similar to templates in C++.
- We can implement
  - Generic classes
  - Generic methods
  - Generic interfaces

Advantages of Generics

- Stronger type checking at compile time i.e. type-safe coding.
- Explicit type casting is not required.
- Generic data structure and algorithm implementation.

Generic Classes

- Implementing a generic class

```java
class Box<TYPE> {
    private TYPE obj;
    public void set(TYPE obj) {
        this.obj = obj;
    }
    public TYPE get() {
        return this.obj;
    }
}
```

```java
Box<String> b1 = new Box<String>();
b1.set("Nilesh");
String obj1 = b1.get();
System.out.println("obj1 : " + obj1);

Box<Date> b2 = new Box<Date>();
b2.set(new Date());
Date obj2 = b2.get();
System.out.println("obj2 : " + obj2);

Box<Integer> b3 = new Box<Integer>();
b3.set(new Integer(11));
String obj3 = b3.get();  // Compiler Error
System.out.println("obj3 : " + obj3);
```

- Instantiating generic class

```
Box<String> b1 = new Box<String>(); // okay

Box<String> b2 = new Box<>(); // okay -- type inference -- type of object is
inferred/guessed looking at reference declaration

Box<> b3 = new Box<>(); // compiler error -- type must be given while
declaring reference

Box<Object> b4 = new Box<String>(); // compiler error

Box b5 = new Box(); // okay -- compiler warning "raw types" -- internally
considered as Object type (for T).
    // not recommended -- doesn't do compiler time type-checking

Box<Object> b6 = new Box<Object>(); // okay -- Not usually required/used
```

Generic types naming convention

1. T : Type
2. N : Number
3. E : Element
4. K : Key
5. V : Value
6. S,U,R : Additional type param

Bounded generic types

- Bounded generic param restricts data type that can be used as type argument.
- Decided by the developer of the generic class.

```
// T can be any type so that T is Number or its sub-class.
class Box<T extends Number> {
    private T obj;
    public T get() {
        return this.obj;
    }
    public void set(T obj) {
        this.obj = obj;
    }
}
```

- The Box<> can now be used only for the classes inherited from the Number class.

```
Box<Number> b1 = new Box<>(); // okay
Box<Boolean> b2 = new Box<>(); // error
Box<Character> b3 = new Box<>(); // error
Box<String> b4 = new Box<>(); // error
```

```
Box<Integer> b5 = new Box<>(); // okay
Box<Double> b6 = new Box<>(); // okay
Box<Date> b7 = new Box<>(); // error
Box<Object> b8 = new Box<>(); // error
```

Unbounded generic types

- Unbounded generic type is indicated with wild-card "?".
- Can be given while declaring generic class reference.

```
class Box<T> {
    private T obj;
    public Box(T obj) {
        this.obj = obj;
    }
    public T get() {
        return this.obj;
    }
    public void set(T obj) {
        this.obj = obj;
    }
}
```

```
public static void printBox(Box<?> b) {
    Object obj = b.get();
    System.out.println("Box contains: " + obj);
}
```

```
Box<String> sb = new Box<String>("DAC");
printBox(sb); // okay
Box<Integer> ib = new Box<Integer>(100);
printBox(ib); // okay
Box<Date> db = new Box<Date>(new Date());
printBox(db); // okay
Box<Float> fb = new Box<Float>(200.5f);
printBox(fb); // okay
```

Upper bounded generic types

- Generic param type can be the given class or its sub-class.

```
public static void printBox(Box<? extends Number> b) {
    Object obj = b.get();
```

```
    System.out.println("Box contains: " + obj);
}
```

```
Box<String> sb = new Box<String>("DAC");
printBox(sb); // error
Box<Integer> ib = new Box<Integer>(100);
printBox(ib); // okay
Box<Date> db = new Box<Date>(new Date());
printBox(db); // error
Box<Object> ob = new Box<Object>(new Object());
printBox(ob); // error
```

Lower bounded generic types

- Generic param type can be the given class or its super-class.

```
public static void printBox(Box<? super Number> b) {
    Object obj = b.get();
    System.out.println("Box contains: " + obj);
}
```

```
Box<String> sb = new Box<String>("DAC");
printBox(sb); // error
Box<Integer> ib = new Box<Integer>(100);
printBox(ib); // error
Box<Object> fb = new Box<Object>(new Object());
printBox(fb); // okay
Box<Number> nb = new Box<Number>(null);
printBox(nb); // okay
```

Generics in Java

Generic Programming = same logic/code for various data types

C language = void*

C++ language = templates

Java (till 1.4) = using Object

Java (5.0 or above) = generics

Generic programming

- generic class
- generic method
- generic interfaces

Generic class

- data structures are idea examples for generic classes
- e.g. stack, queue, linked list, ...

JVM architecture

OS process for "java" (java app launcher)

```java
// predefined class
class HttpServlet ... {
    public void init(ServletConfig cfg) throws ServletException {
        // ...
    }
}

class MyServlet extends HttpServlet {
    @Override
    public void init(ServletConfig cfg) throws ServletException {
        try {
            // database connection
        } catch(SQLException ex) {
            throw new ServletException(ex); // exception chaining
        }
    }
}
```

2)

1.try with resource

```
try(Scanner sc = new Scanner(System.in)){
    //..
} // sc.close();
```

2.try with catch

```
try{
 //..
}catch(Exception e) {
 //..
}
```

3.try with finally

```
try{
  //...
}
finally{
  //...
}
```

4.try with catch and finally

```
try{
 //..
}
catch(Exception e){
 // ...
}
finally{
 //...
}
```

day10 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
        System.out.println("Result: " + result);
    }
    */

    // detect the problem and throw the problem
    public static int divide(int num, int den) {
        if(den == 0)
            throw new RuntimeException("Cannot Divide by Zero.");
        int res = num / den;
        return res;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter numerator and denominator: ");
        int num = sc.nextInt();
        int den = sc.nextInt();
        try {
            int result = divide(num, den);
            System.out.println("Result: " + result);
        }
        catch (RuntimeException e) {
            System.out.println("Divide Error");
        }
    }
}
```

30 6 5 30 6 5

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclip

```
Enter numerator and denominator: 30 6
Result: 5
```

Search 11:52 AM

day10 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```
38          System.out.println("Result: " + result);
39      }
40      */
41
42      // detect the problem and throw the problem
43      public static int divide(int num, int den) {
44          if(den == 0)                    22      0
45              throw new RuntimeException("Cannot Divide by Zero.");
46          int res = num / den;
47          return res;                     xxx not executed
48      }
49      public static void main(String[] args) {
50          Scanner sc = new Scanner(System.in);
51          System.out.print("Enter numerator and denominator: ");
52          int num = sc.nextInt(); 22
53          int den = sc.nextInt(); 0
54          try {
55              int result = divide(num, den);
56              System.out.println("Result: " + result); xxx not executed
57          }
58          catch (RuntimeException e) {
59              System.out.println("Divide Error");
60          }
61      }
62  }
63
```

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclip

```
Enter numerator and denominator: 22 0
Divide Error
```

Search 11:53 AM

day10 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
        System.out.println("Result: " + result);
    }
    */

    // detect the problem and throw the problem
    public static int divide(int num, int den) {
        if(den == 0)
            throw new RuntimeException("Cannot Divide
        int res = num / den;
        return res;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
//      try {
            System.out.print("Enter numerator and denominator: ");
            int num = sc.nextInt();
            int den = sc.nextInt();
            int result = divide(num, den);
            System.out.println("Result: " + result);
//      }
//      catch (RuntimeException e) {
//          System.out.println("Divide Error");
//      }
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.

```
Enter numerator and denominator: Four Two
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at com.sunbeam.Program02.main(Program02.java:53)
```

Search 12:18 PM

day10 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```
39          System.out.println("Result: " + resul
40      }
41      */
42
43      // detect the problem and throw the probl
44      public static int divide(int num, int den
45          int res = num / den;
46          return res;
47      }
48      public static void main(String[] args) {
49          Scanner sc = new Scanner(System.in);
50          try {
51              System.out.print("Enter numerator and denominator: ");
52              int num = sc.nextInt();
53              int den = sc.nextInt();
54              int result = divide(num, den);
55              System.out.println("Result: " + result);
56          }
57          catch (InputMismatchException e) {
58              System.out.println("Invalid User Input.");
59          }
60 //       catch (RuntimeException e) {
61 //           System.out.println("Divide Error");
62 //       }
63      }
64 }
```

Problems | Javadoc | Declaration | Console

```
<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1
Enter numerator and denominator: 22 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at com.sunbeam.Program02.divide(Program02.java:45)
        at com.sunbeam.Program02.main(Program02.java:54)
```

Java division (/) operator internally checks denominator and if it is zero, it will throw the exception -- ArithmeticException.

Search 12:20 PM

day10 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
        System.out.println("Result: " + result);
    }
    */

    // detect the problem and throw the problem
    public static int divide(int num, int den) {
        int res = num / den;
        return res;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter numerator and denominator: ");
            int num = sc.nextInt();
            int den = sc.nextInt();
            int result = divide(num, den);
            System.out.println("Result: " + result);
        }
        catch (InputMismatchException e) {
            System.out.println("Invalid User Input.");
        }
        catch (ArithmeticException e) {
            System.out.println("Divide Error");
        }
    }
}
```

try with multiple catch blocks.

multiple try with single/common catch is not possible.

Writable | Smart Insert | 63 : 6 : 1724

Search 12:21 PM

day10 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
            System.out.println("Divide Error");
        }
    }
    */

    // detect the problem and throw the problem
    public static int divide(int num, int den) {
        int res = num / den; // divide operator may throw Arithmetic
        return res;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // we can write common catch block for multiple exceptions
        try {
            System.out.print("Enter numerator and denominator: ");
            int num = sc.nextInt(); // may throw InputMismatchException
            int den = sc.nextInt();
            int result = divide(num, den);
            System.out.println("Result: " + result);
        }
        catch (InputMismatchException | ArithmeticException e) {
            System.out.println("Some Problem Occurred.");
        }
    }
}
```

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse

```
Enter numerator and denominator: 22 0
Some Problem Occurred.
```

Search 12:30 PM

day10 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
            System.out.println("Divide Error");
        }
    }
    */

    // detect the problem and throw the problem
    public static int divide(int num, int den) {
        int res = num / den; // divide operator may throw Arithmetic
        return res;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // we can write common catch block for multiple exceptions
        try {
            System.out.print("Enter numerator and denominator: ");
            int num = sc.nextInt(); // may throw InputMismatchException
            int den = sc.nextInt();
            int result = divide(num, den);
            System.out.println("Result: " + result);
        }
        catch (InputMismatchException | ArithmeticException e) {
            System.out.println("Some Problem Occurred.");
        }
    }
}
```

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse

```
Enter numerator and denominator: Four Two
Some Problem Occurred.
```

Search 12:30 PM

day10 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java

    // detect the problem and throw the problem
    public static int divide(int num, int den) {
        int res = num / den; // divide operator may throw ArithmeticException
        return res;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // we can write common catch block for multiple exceptions
        try {
            System.out.print("Enter numerator and denominator: ");
            int num = sc.nextInt(); // may throw InputMismatchException
            int den = sc.nextInt();
            int result = divide(num, den);
            System.out.println("Result: " + result);
            //sc.close(); // will not execute if exception occurrs
        }
        catch (InputMismatchException | ArithmeticException e) {
            //sc.close(); // will not execute if exception doesn't occur
            System.out.println("Some Problem Occurred.");
        }
        finally {
            sc.close(); // will execute irrespective of exception
            // alternate option is to use try-with-resource -- learned yesterday
        }
    }
```

Writable | Smart Insert | 132 : 16 [7]

Search 12:35 PM

day09.md | day10.md | classwork.md U | Untitled-1

day10.md > # Core Java > ## Garbage collection

```
* Java exception class hierarchy
  ```
  Object
      |- Throwable
          |- Error
          | |- AssertionError
          | |- VirtualMachineError
          | |- StackOverflowError
          | |- OutOfMemoryError
          |- Exception
              |- CloneNotSupportedException
              |- IOException
              | |- EOFException
              | |- FileNotFoundException
              |- SQLException
              |- InterruptedException
              |- RuntimeException
                  |- NullPointerException
                  |- ArithmeticException
                  |- NoSuchElementException
                  | |- InputMismatchException
                  |- IndexOutOfBoundsException
                          |- ArrayIndexOutOfBoundsException
                          |- StringIndexOutOfBoundsException
  ```
```

Error: Usually represents problems that are not recoverable.
Typical handling -- display error and System.exit().

Exception: Usually represents problems that are recoverable.
Typical handling -- catch -- handling logic.

Checked Exceptions
- Checked by the Java compiler.
- Programmer must be aware of/ready for these exceptions.
  - implement catch block
  or - throws clause to send error to the calling method
- Most of checked exceptions arise out of JVM
- all classes inherited from Exception (except RuntimeEx...).

Unchecked Execptions
- Not checked by Java compiler
- Programmer may or may not handle the exception.
  - If not handled, JVM will catch and terminate program.
- Compiler do not force to write catch block or throws clause.
- All classes inherited from RuntimeException.
- Most of these exception occurrs within JVM due to Programmer's/User's mistake.b

main* | 0 0 | 0 | Ln 55, Col 1 (78 selected) | Spaces: 4 | UTF-8 | CRLF | Markdown

day11 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java

```java
package com.sunbeam;

import java.util.Scanner;

public class Program01 {
    public static int divide(int num, int den) {
        return num / den;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter two numbers: ");
        int n = sc.nextInt();
        int d = sc.nextInt();
        int r = divide(n, d);
        System.out.println("Result: " + r);
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 12, 2024, 9:51:12 AM – 9:51:18 AM) [pid: 12772]

```
Enter two numbers: 22 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at com.sunbeam.Program01.divide(Program01.java:7)
        at com.sunbeam.Program01.main(Program01.java:14)
```

exception class

message

stack trace

Writable | Smart Insert | 14 : 30 : 351

Search | 9:51 AM

day11 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java

```java
import java.util.Scanner;

public class Program01 {
    public static int divide(int num, int den) {
        return num / den;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter two numbers: ");
            int n = sc.nextInt();
            int d = sc.nextInt();
            int r = divide(n, d);
            System.out.println("Result: " + r);
        }
        catch(Throwable e) {
            System.out.println("Exception Message: " + e.getMessage());
        }
    }
```

Problems | Javadoc | Declaration | Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 12, 2024, 9:56:17 AM – 9:56:20 AM) [pid: 12220]

```
Enter two numbers: 22 0
Exception Message: / by zero
```

Search 9:56 AM

day11 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java

```java
    public static int divide(int num, int den) {
        return num / den;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter two numbers: ");
            int n = sc.nextInt();
            int d = sc.nextInt();
            int r = divide(n, d);
            System.out.println("Result: " + r);
        }
        catch(Throwable e) {
            //System.out.println("Exception Message: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 12, 2024, 9:57:52 AM – 9:57:54 AM) [pid: 1360]

```
Enter two numbers: 22 0
java.lang.ArithmeticException: / by zero
        at com.sunbeam.Program01.divide(Program01.java:7)
        at com.sunbeam.Program01.main(Program01.java:15)
```

Search 9:57 AM

day11 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java

```java
    public static int divide(int num, int den) {
        return num / den;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter two numbers: ");
            int n = sc.nextInt();
            int d = sc.nextInt();
            int r = divide(n, d);
            System.out.println("Result: " + r);
        }
        catch(Throwable e) {
            //System.out.println("Exception Message: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Throwable

e → ArithmeticException

up-casting

generic catch block:

Exception of any type can be handle in a super-exception type reference (like Throwable, Error, Exception, or RuntimeException as appropriate).

Problems | Javadoc | Declaration | Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 12, 2024, 9:57:52 AM – 9:57:54 AM) [pid: 1360]

```
Enter two numbers: 22 0
java.lang.ArithmeticException: / by zero
        at com.sunbeam.Program01.divide(Program01.java:7)
        at com.sunbeam.Program01.main(Program01.java:15)
```

Search 9:58 AM

day11 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java

```java
public class Program03 {
    public static void main(String[] args) {
        Box b1 = new Box();
        b1.set(new Double(3.14));
        Double r1 = (Double) b1.get();
        System.out.println("r1 = " + r1);

        Box b2 = new Box();
        b2.set(new Date());
        Date r2 = (Date) b2.get();
        System.out.println("r2 = " + r2);

        Box b3 = new Box();
        b3.set("123");
        Integer r3 = (Integer) b3.get();
        System.out.println("r3 = " + r3);
    }
}
```

type-checking is done by JVM at runtime.
-- Not Typesafe.

Problems | Javadoc | Declaration | Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 12, 2024, 11:46:13 AM – 11:46:14 AM) [pid: 1452(

```
r1 = 3.14
r2 = Mon Feb 12 11:46:14 IST 2024
Exception in thread "main" java.lang.ClassCastException: class java.lang.String cannot be cast to class java.lang.Integer (ja
        at com.sunbeam.Program03.main(Program03.java:30)
```

Search 11:46 AM

day11 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- > demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program01.java
- > demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - InvalidTimeException.ja
      - Program02.java
      - Time.java
- > demo03 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program03.java
- > demo04 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program04.java

Program03.java | Program04.java

```java
public class Program04 {
    public static void main(String[] args) {
        Box<Double> b1 = new Box<Double>();
        b1.set(new Double(3.14));
        Double r1 = b1.get();
        System.out.println("r1 = " + r1);

        Box<Date> b2 = new Box<Date>();
        b2.set(new Date());
        Date r2 = b2.get();
        System.out.println("r2 = " + r2);

        Box<String> b3 = new Box<String>();
        b3.set("123");
        //Integer r3 = b3.get(); // compiler error: compile time type checking = type-safety
        String r3 = b3.get();
        System.out.println("r3 = " + r3);
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 12, 2024, 11:5

```
r1 = 3.14
r2 = Mon Feb 12 11:51:43 IST 2024
r3 = 123
```

Search | 11:51 AM

day11 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- > demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program01.java
- > demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - InvalidTimeException.ja
      - Program02.java
      - Time.java
- > demo03 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program03.java
- > demo04 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program04.java

Program03.java × Program04.java

```
31          b3.set("123");
32          Integer r3 = (Integer) b3.get();
33          System.out.println("r3 = " + r3);
34      }
35      */
36      public static void main(String[] args) {
37          // Java collections -- type-safety problem -- till Java 1.4
38          ArrayList list = new ArrayList<>();
39          list.add(11);
40          list.add(22);
41          list.add(33);
42          list.add("44");
43          for(Object ele : list) {
44              Integer num = (Integer) ele;
45              System.out.println(num);
46          }
47      }
48  }
49
```

no type-check at compile time (can stote any Object

runtime type-check -- ClassCastException

Problems Javadoc Declaration Console ×

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 12, 2024, 11:5

```
11
22
33
Exception in thread "main" java.lang.ClassCastException: class java.lang.String cannot be cast to class ja
```

Writable | Smart Insert | 39 : 22 : 864

Search 11:54 AM

day11 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- > demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program01.java
- > demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - InvalidTimeException.ja
      - Program02.java
      - Time.java
- > demo03 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program03.java
- > demo04 [CJ-H-02 main]
  - JRE System Library [JavaSE-1
  - > src
    - > com.sunbeam
      - Program04.java

Program03.java | Program04.java

```java
        b3.set("123");
        //Integer r3 = b3.get(); // compiler error: compile time type checking = type-safety
        String r3 = b3.get();
        System.out.println("r3 = " + r3);
    }
    */
    public static void main(String[] args) {
        // Java collections -- type-safety problem -- till Java 1.4
        ArrayList<Integer> list = new ArrayList<Integer>();
        list.add(11);
        list.add(22);
        list.add(33);
        //list.add("44"); // -- compiler error -- compile time type-checking = type-safety
        for(Integer ele : list) {
            System.out.println(ele);
        }
    }

}
```

Problems | Javadoc | Declaration | Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 12, 2024, 11:5

```
11
22
33
```

Writable | Smart Insert | 43 : 91 [82]

Search | 11:57 AM

day11 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java × Program04.java

```java

public class Program05 {
    public static void main(String[] args) {
        Box<Integer> b1 = new Box<>(); // Integer is-a Number
        b1.set(new Integer(123));
        System.out.println("Integer: " + b1.get());

        Box<Double> b2 = new Box<>(); // Double is-a Number
        b2.set(new Double("3.14"));
        System.out.println("Double: " + b2.get());

        Box<Number> b3 = new Box<>(); // Number
        b3.set(new Long(1));
        System.out.println("Number: " + b3.get());

        //Box<String> b4 = new Box<>(); // compiler error: String is not Number
        //Box<Boolean> b5 = new Box<>(); // compiler error: Boolean is not Number
        //Box<Object> b4 = new Box<>(); // compiler error: Object is not Number
    }
```

Problems Javadoc Declaration Console ×

<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 12, 2024, 12:09:30 PM – 12:09:30 PM) [pid: 15452

```
Integer: 123
Double: 3.14
Number: 1
```

Writable | Smart Insert | 27 : 9 : 677

Search 12:09 PM

day11 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java × Program04.java

```java
class Box<T extends Number> {
    private T obj;
    public void set(T obj) {
        this.obj = obj;
    }
    public T get() {
        return this.obj;
    }
}

public class Program05 {
    public static void main(String[] args) {
        Box<Integer> b1 = new Box<>(); // Integer is-a Number
        b1.set(new Integer(123));
        System.out.println("Integer: " + b1.get());

        Box<Double> b2 = new Box<>(); // Double is-a Number
        b2.set(new Double("3.14"));
        System.out.println("Double: " + b2.get());

        Box<Number> b3 = new Box<>(); // Number
        b3.set(new Long(1));
        System.out.println("Number: " + b3.get());

        //Box<String> b4 = new Box<>(); // compiler error: String is not Number
        //Box<Boolean> b5 = new Box<>(); // compiler error: Boolean is not Number
```

Writable | Smart Insert | 15 : 32 : 274

Search 12:11 PM

```
    Box<Integer> b3 = new Box<Integer>();
    b3.set(new Integer(11));
    String obj3 = b3.get();  // Compiler Error
    System.out.println("obj3 : " + obj3);
    ```
* Instantiating generic class
    ```Java
    Box<String> b1 = new Box<String>(); // okay

    Box<String> b2 = new Box<>(); // okay -- type inference -- type of object is inferred/guessed looking at reference declaration

    Box<> b3 = new Box<>(); // compiler error -- type must be given while declaring reference

    Box<Object> b4 = new Box<String>(); // compiler error

    Box b5 = new Box(); // ? -- compiler warning "raw types"

    Box<Object> b6 = new Box<Object>(); // ? -- Not usually required/used
    ```

Generic types naming convention
1. T : Type
```

Object ↑ is-a String

Box<Object> ↑ ✗ is not Box<String>
```

Core Java

Generic Programming

Generic Methods

- Generic methods are used to implement generic algorithms.
- Example:

```
// non type-safe
void printArray(Object[] arr) {
    for(Object ele : arr)
        System.out.println(ele);
    System.out.println("Number of elements printed: " + arr.length);
}
```

```
// type-safe
<T> void printArray(T[] arr) {
    for(T ele : arr)
        System.out.println(ele);
    System.out.println("Number of elements printed: " + arr.length);
}
```

```
String[] arr1 = { "John", "Dagny", "Alex" };
printArray(arr1); // printArray<String> -- String type is inferred

Integer[] arr2 = { 10, 20, 30 };
printArray(arr2); // printArray<Integer> -- Integer type is inferred
```

Generics Limitations

1. Cannot instantiate generic types with primitive Types. Only reference types are allowed.

```
ArrayList<Integer> list = new ArrayList<Integer>(); // okay
ArrayList<int> list = new ArrayList<int>(); // compiler error
```

2. Cannot create instances of Type parameters.

```
Integer i = new Integer(11); // okay
T obj = new T(); // error
```

3. Cannot declare static fields with generic type parameters.

```
class Box<T> {
    private T obj; // okay
    private static T object; // compiler error
    // ...
}
```

4. Cannot Use casts or instanceof with generic Type params.

```
if(obj instanceof T) {  // compiler error
    newobj = (T)obj;    // compiler error
}
```

5. Cannot Create arrays of generic parameterized Types

```
T[] arr = new T[5]; // compiler error
```

6. Cannot create, catch, or throw Objects of Parameterized Types

```
throw new T(); // compiler error

try {
    // ...
} catch(T ex) { // compiler error
    // ...
}
```

7. Cannot overload a method just by changing generic type. Because after erasing/removing the type param, if params of two methods are same, then it is not allowed.

```
public void printBox(Box<Integer> b) {
    // ...
}
public void printBox(Box<String> b) { // compiler error
    // ...
}
```

Type erasure

- The generic type information is erased (not maintained) at runtime (in JVM). `Box<Integer>` and `Box<Double>` both are internally (JVM level) treated as Box objects. The field "T obj" in Box class, is

treated as "Object obj".

- Because of this method overloading with genric type difference is not allowed.

```
void printBox(Box<Integer> b) { ... }
    // void printBox(Box b) { ... } <-- In JVM
void printBox(Box<Double> b) { ... } //compiler error
    // void printBox(Box b) { ... } <-- In JVM
```

Generic Interfaces

- Interface is standard/specification.

```
// Comparable is pre-defined interface -- non-generic till Java 1.4
interface Comparable {
    int compareTo(Object obj);
}
class Person implements Comparable {
    // ...
    public int compareTo(Object obj) {
        Person other = (Person)obj; // down-casting
        // compare "this" with "other" and return difference
    }
}
class Program {
    public static void main(String[] args) {
        Person p1 = new Person("James Bond", 50);
        Person p2 = new Person("Ironman", 45);
        int diff = p1.compareTo(p2);
        if(diff == 0)
            System.out.println("Both are same");
        else if(diff > 0)
            System.out.println("p1 is greater than p2");
        else //if(diff < 0)
            System.out.println("p1 is less than p2");

        diff = p2.compareTo("Superman"); // will fail at runtime with
ClassCastException (in down-casting)
    }
}
```

- Generic interface has type-safe methods (arguments and/or return-type).

```
// Comparable is pre-defined interface -- generic since Java 5.0
interface Comparable<T> {
    int compareTo(T obj);
}
class Person implements Comparable<Person> {
    // ...
```

```
    public int compareTo(Person other) {
        // compare "this" with "other" and return difference


    }
}
class Program {
    public static void main(String[] args) {
        Person p1 = new Person("James Bond", 50);
        Person p2 = new Person("Ironman", 45);
        int diff = p1.compareTo(p2);
        if(diff == 0)
            System.out.println("Both are same");
        else if(diff > 0)
            System.out.println("p1 is greater than p2");
        else //if(diff < 0)
            System.out.println("p1 is less than p2");

        diff = p2.compareTo("Superman"); // compiler error
    }
}
```

Comparable<>

- Standard for comparing the current object to the other object.
- Also referred as "Natural Ordering" for the class.
- Has single abstract method `int compareTo(T other);`
- In java.lang package.
- Used by various methods like Arrays.sort(Object[]), ...

```
// pre-defined interface
interface Comparable<T> {
    int compareTo(T other);
}
```

```
class Employee implements Comparable<Employee> {
    private int empno;
    private String name;
    private int salary;
    // ...
    public int compareTo(Employee other) {
        int diff = this.empno - other.empno;
        return diff;
    }
}
```

```
Employee e1 = new Employee(2, "Sarang", 50000);
Employee e2 = new Employee(1, "Nitin", 40000);
int diff = e1.compareTo(e2);
```

```
Employee[] arr = { ... };
Arrays.sort(arr);
for(Employee e:arr)
    System.out.println(e);
```

Comparator<>

- Standard for comparing two (other) objects.
- Has single abstract method `int compare(T obj1, T obj2);`
- In java.util package.
- Used by various methods like Arrays.sort(T[], comparator), ...

```
// pre-defined interface
interface Comparator<T> {
    int compare(T obj1, T obj2);
}
```

```
class EmployeeSalaryComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        if(e1.getSalary() == e2.getSalary())
            return 0;
        if(e1.getSalary() > e2.getSalary())
            return +1;
        return -1;
    }
}
```

Multi-level sorting

```
class Employee implements Comparable<Employee> {
    private int empno;
    private String name;
    private String designation;
    private int department;
    private int salary;
```

```
    // ...
}
```

```
// Multi-level sorting -- 1st level: department, 2nd level: designation, 3rd
level: salary(int)
class CustomComparator implements Comparator<Employee> {
    public int compare(Employee e1, Employee e2) {
        int diff = e1.getDepartment().compareTo(e2.getDepartment());
        if(diff == 0)
            diff = e1.getDesignation().compareTo(e2.getDesignation());
        if(diff == 0)
            diff = e1.getSalary() - e2.getSalary();
        return diff;
    }
}
```

```
Employee[] arr = { ... };
Arrays.sort(arr, new CustomComparator());
// ...
```

Java Collection Framework

- Collection framework is Library of reusable data structure classes that is used to develop application.
- Main purpose of collection framework is to manage data/objects in RAM efficiently.
- Collection framework was introduced in Java 1.2 and type-safe implementation is provided in 5.0 (using generics).
- java.util package.
- Java collection framework provides
    - Interfaces -- defines standard methods for the collections.
    - Implementations -- classes that implements various data stuctures.
    - Algorithms -- helper methods like searching, sorting, ...

Collection Hierarchy

- Interfaces: Iterable, Collection, List, Queue, Set, Map, Deque, SortedSet, SortedMap, ...
- Implementations: ArrayList, LinkedList, HashSet, HashMap, ...
- Algorithms: sort(), reverse(), max(), min(), ... -> in Collections class static methods

Iterable interface

- To traverse any collection it provides an Iterator.
- Enable use of for-each loop.
- In java.lang package
- Methods
    - Iterator iterator() // SAM

    - default Spliterator spliterator()
    - default void forEach(Consumer<? super T> action)

Collection interface

- Root interface in collection framework interface hierarchy.
- Most of collection classes are inherited from this interface (indirectly).
- Provides most basic/general functionality for any collection
- Abstract methods
    - boolean add(E e)
    - int size()
    - boolean isEmpty()
    - void clear()
    - boolean contains(Object o)
    - boolean remove(Object o)
    - boolean addAll(Collection<? extends E> c)
    - boolean containsAll(Collection<?> c)
    - boolean removeAll(Collection<?> c)
    - boolean retainAll(Collection<?> c)
    - Object[] toArray()
    - Iterator iterator() -- inherited from Iterable
- Default methods
    - default Stream stream()
    - default Stream parallelStream()
    - default boolean removeIf(Predicate<? super E> filter)

Assignment

1. Write a generic static method to find minimum from an array of Number.
2. Use Arrays.sort() to sort array of Students using Comparator. The 1st level sorting should be on city (desc), 2nd level sorting should be on marks (desc), 3rd level sorting should be on name (asc).

```
class Student {
    private int roll;
    private String name;
    private String city;
    private double marks;
    // ...
}
```

Generic Methods

```
Static void swap(Object x, Object y){
    Object t=x;
    x=y;
    y=t;
    Sysout("x="+x+", y="+y);
}

Static void main(String[] args){
    String s1="A", s2="B";
    Swap(s1, s2);
      //x=B, y=A
    Integer i=12;
    Double d=3.14;
    Swap(i, d);
      // x=3.14, y=12
}
```

```
Static <T> void swap(T x, T y){
    T t=x;
    x=y;
    y=t;
    Sysout("x="+x+", y="+y);
}
Static void main(String[] args){
    String s1="A", s2="B";
    Swap(s1, s2); //T=String
      //x=B, y=A
    Integer i=12;
    Double d=3.14;
    Swap(i, d); //T=Number
      // x=3.14, y=12
  ClsName.<Double>Swap(i, d); //T=Double
}
```

type inference

type given manually

example → Compiler error: Integer ≠ Double

```
//pre-defined class
class Object {
    //...
    public boolean equals(Object o);
}

class Fraction extends Object {
    private int n,d;
    //...
    public boolean equals(Object o) {
        //...
        Fraction other=(Fraction) o;
        if(n==other.n && d==other.d)
            return true;
        return false;
    }
}
```

```
main()
    Fraction f1=new Fraction(...);
    f1.equals("22/7");
```

Comparable

Comparable is standard for comparing "this" object with the given "other" object.

```
// pre-defined interface (1.4)
interface Comparable {
    int compareTo(Object o);
}
class Fraction implements Comparable {
    private int n, d;
    public double value() {
        return (double) n/d;
    }
    @override
    public int compareTo(Object o) {
        Fraction other=(Fraction) o;
        if(this.value() == other.value())
            return 0;
        if (this.value() > other.value())
            return +1;
        else return -1;
    }
}
```

compareTo() method should return diff between <u>this</u> and <u>other</u> object.

0 → if this == other
+ve → if this > other
-ve → if this < other

```
main();
    Fraction f1=new Fraction(10, 2);
    Fraction f2=new Fraction(20, 2);
    diff = f1.compareTo(f2);
        ↳ -1 (f1 < f2)
    diff = f1.compareTo("9/3");
        ↳ ClassCastException
```

Comparable<T>

Comparable is standard for comparing "this" object with the given "other" object.

```
// pre-defined interface (5.0)
interface Comparable<T> {
    int compareTo(T o);
}

class Fraction implements Comparable<Fraction> {
    private int n, d;
    public double value() {
        return (double) n/d;
    }
    @override
    public int compareTo(Fraction o) {
        if(this.value() == o.value())
            return 0;
        if (this.value() > o.value())
            return +1;
        else
            return -1;
    }
}
```

compareTo() method should return diff between <u>this</u> and <u>other</u> object.

0 → if this == other
+ve → if this > other
-ve → if this < other

```
main();
    Fraction f1=new Fraction(10, 2);
    Fraction f2=new Fraction(20, 2);
    diff = f1.compareTo(f2); ✓
      ↳ -1 (f1 < f2)
    diff = f1.compareTo("9/3");
                          ✗
                Compiler error
```

Java Collection Framework

Java 1.0 → Limited collection classes
- ✓ Vector, Hashtable, Stack, ...

Java 1.2 → Collection Framework
- ✓ efficient data processing
  - space & time
- ✓ Collection framework
  - interfaces
  - implementations
  - algorithms

java.util Package

Java collection framework

Collection: basic collection i.e. add ele, remove ele, traverse collection, etc.

List: allow duplicate, random access, bi-direction traversal, searching.

Set: allow unique elems only

Queue: utility ds like stack & queue

Map: key value pairs for fast searching.

Since Java 5.0

```
class Box <TYPE> {
    private TYPE obj;
    public void set(TYPE obj) {
        this.obj = obj;
    }
    public TYPE get() {
        return this.obj;
    }
}
```

```
main():
    Box<String> b1 = new Box<String>();
    b1.set("Hello");
    String r1 = b1.get();
    Box<Double> b2 = new Box<Double>();
    b2.set(3.14);
    Double r2 = b2.get();
```

Java Compiler → .class file → JVM

```
class Box {
    private Object obj;
    public void set(Object obj) {
        this.obj = obj;
    }
    public Object get() {
        return this.obj;
    }
}
```

The type-safety of Java generics is ensured by the compiler. JVM doesn't do any type-checking at runtime.

For JVM all references are like Object references.

There is no type info present in .class file. This called as Type Erasure.

```Java
// T can be any type so that T is Number or its sub-class.
class Box<T extends Number> {
    private T obj;
    public T get() {
        return this.obj;
    }
    public void set(T obj) {
        this.obj = obj;
    }
}
```

* The Box<> can now be used only for the classes inherited from the Number class.

```Java
Box<Number> b1 = new Box<>(); // okay
Box<Boolean> b2 = new Box<>(); // error
Box<Character> b3 = new Box<>(); // error
Box<String> b4 = new Box<>(); // error
Box<Integer> b5 = new Box<>(); // okay
Box<Double> b6 = new Box<>(); // okay
Box<Date> b7 = new Box<>(); // error
Box<Object> b8 = new Box<>(); // error
```

```Java
interface Shape {
    // ...
}
class Circle implements Shape {
    // ...
}
class Rectangle implements Shape {
    // ...
}
```

Syntax is Valid.

Use of implements is not allowed in < ... >. Use extends.

```Java
class Box<T extends Shape> {
    T obj;
    T get() { return this.obj; }
    void set(T obj) { this.obj = obj; }
}
```

In main():

```Java
Box<Circle> b1 = new Box<>();
b1.set(new Circle());
```

day12 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Product1.java | Program02.java

```
9          System.out.println("Before Sort: " + Arrays.toString(arr));
10         Arrays.sort(arr);
11         System.out.println(" After Sort: " + Arrays.toString(arr));
12     }
13     */
14
15     public static void main(String[] args) {
16         Product1[] arr = {
17             new Product1(3, "Pen", 45.0),
18             new Product1(1, "Pencil", 5.0),
19             new Product1(2, "Eraser", 3.0),
20             new Product1(5, "Paper", 6.0),
21             new Product1(4, "Notebook", 80.0)
22         };
23         System.out.println("Before Sort:");
24         for (int i = 0; i < arr.length; i++)
25             System.out.println(arr[i]);
26
27         Arrays.sort();  Arrays.sort(arr);
28
29         System.out.println(" After Sort:");
30         for (int i = 0; i < arr.length; i++)
31             System.out.println(arr[i]);
32     }
33 }
34
```

Writable | Smart Insert | 27 : 23 : 705

Arrays.sort() internally use quick-sort for sorting elements and internally calls Comparable.compareTo() on array elements if and when elements need to be compared.

```
Object                    Object
<TYPE>void selectionSort( TYPE [] arr) {
    for(int i=0; i<arr.length-1; i++) {
        for(int j=i+1; j<arr.length; j++) {
            if(arr[i] > arr[j]) { a[i].compareTo(a[j]) > 0
                Object TYPE t = arr[i];
                arr[i] = arr[j];
                arr[j] = t;
            }
        }
    }
}
```

to make selectionSort() generic, use some std for comparing arr[i] and arr[j].

e.g. Comparable, Comparator

Search | 10:31 AM

day12 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Product1.java | Program02.java

```java
        System.out.println("Before Sort: " + Arrays.toString(arr));
        Arrays.sort(arr);
        System.out.println(" After Sort: " + Arrays.toString(arr));
    }
    */

    public static void main(String[] args) {
        Product1[] arr = {
            new Product1(3, "Pen", 45.0),
            new Product1(1, "Pencil", 5.0),
            new Product1(2, "Eraser", 3.0),
            new Product1(5, "Paper", 6.0),
            new Product1(4, "Notebook", 80.0)
        };
        System.out.println("Before Sort:");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);

        Arrays.sort(arr);
```

Yet, Product1 class is not inherited from Comparable.

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 13, 2024, 10:41:44 AM – 10:41:44 AM) [pid: 1834

```
id=5, name=Paper, price=6.0]
id=4, name=Notebook, price=80.0]
in thread "main" java.lang.ClassCastException: class com.sunbeam.Product1 cannot be cast to class java.lang.Comparable (com.s
 java.base/java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:320)
 java.base/java.util.ComparableTimSort.sort(ComparableTimSort.java:188)
```

Search 10:41 AM

day12 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Product1.java | Program02.java

```java
        System.out.println("Before Sort: " + Arrays.toString
        Arrays.sort(arr);
        System.out.println(" After Sort: " + Arrays.toString
    }
    */

    public static void main(String[] args) {
        Product1[] arr = {
            new Product1(3, "Pen", 45.0),
            new Product1(1, "Pencil", 5.0),
            new Product1(2, "Eraser", 3.0),
            new Product1(5, "Paper", 6.0),
            new Product1(4, "Notebook", 80.0)
        };
        System.out.println("Before Sort:");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);

        Arrays.sort(arr);

        System.out.println(" After Sort:");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hc

```
Before Sort:
Product1 [id=3, name=Pen, price=45.0]
Product1 [id=1, name=Pencil, price=5.0]
Product1 [id=2, name=Eraser, price=3.0]
Product1 [id=5, name=Paper, price=6.0]
Product1 [id=4, name=Notebook, price=80.0]
 After Sort:
Product1 [id=1, name=Pencil, price=5.0]
Product1 [id=2, name=Eraser, price=3.0]
Product1 [id=3, name=Pen, price=45.0]
Product1 [id=4, name=Notebook, price=80.0]
Product1 [id=5, name=Paper, price=6.0]
```

Product1 class inherited from Comparable and comparison is done on "id".

Search 10:44 AM

day12 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

```
        System.out.println(" After Sort:")
        for (int i = 0; i < arr.length; i+
            System.out.println(arr[i]);
    }
    */

    public static void main(String[] args)
        Product2[] arr = {
            new Product2(3, "Pen", 45.0),
            new Product2(1, "Pencil", 5.0)
            new Product2(2, "Eraser", 3.0)
            new Product2(5, "Paper", 6.0),
            new Product2(4, "Notebook", 80
        };
        System.out.println("Before Sort:");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);

        Arrays.sort(arr);

        System.out.println(" After Sort:");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

```
<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hc
Product2 [id=3, name=Pen, price=45.0]
Product2 [id=1, name=Pencil, price=5.0]
Product2 [id=2, name=Eraser, price=3.0]
Product2 [id=5, name=Paper, price=6.0]
Product2 [id=4, name=Notebook, price=80.0]
 After Sort:
Product2 [id=2, name=Eraser, price=3.0]
Product2 [id=4, name=Notebook, price=80.0]
Product2 [id=5, name=Paper, price=6.0]
Product2 [id=3, name=Pen, price=45.0]
Product2 [id=1, name=Pencil, price=5.0]
```

Product2 class inherited from Comparable and comparison is done on "name".

day12 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

```java
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
    */

    public static void main(String[] args) {
        Product3[] arr = {
            new Product3(3, "Pen", 45.0),
            new Product3(1, "Pencil", 5.0),
            new Product3(2, "Eraser", 3.0),
            new Product3(5, "Paper", 6.0),
            new Product3(4, "Notebook", 80.0)
        };
        System.out.println("Before Sort:");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);

        Arrays.sort(arr);

        System.out.println(" After Sort:");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }

}
```

```
<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hc
Before Sort:
Product3 [id=3, name=Pen, price=45.0]
Product3 [id=1, name=Pencil, price=5.0]
Product3 [id=2, name=Eraser, price=3.0]
Product3 [id=5, name=Paper, price=6.0]
Product3 [id=4, name=Notebook, price=80.0]
 After Sort:
Product3 [id=2, name=Eraser, price=3.0]
Product3 [id=1, name=Pencil, price=5.0]
Product3 [id=5, name=Paper, price=6.0]
Product3 [id=3, name=Pen, price=45.0]
Product3 [id=4, name=Notebook, price=80.0]
```

class Product3 inherited from Comparable and comparison done by "price" in asc order.

day12 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Product1.java | Program02.java | Product2.java | Product3.java

```java
package com.sunbeam;

import java.util.Arrays;

public class Program02 {
    /*
    public static void main(String[] args) {
        int[] arr = { 33, 66, 22, 55, 44 };
        System.out.println("Before Sort: " + Arrays.toString(arr));
        Arrays.sort(arr);
        System.out.println(" After Sort: " + Arrays.toString(arr));
    }
    */

    /*
    public static void main(String[] args) {
        Product1[] arr = {
            new Product1(3, "Pen", 45.0),
            new Product1(1, "Pencil", 5.0),
            new Product1(2, "Eraser", 3.0),
            new Product1(5, "Paper", 6.0),
            new Product1(4, "Notebook", 80.0)
        };
        System.out.println("Before Sort:");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
```

Comparable = Natural Ordering

-- in-built ordering i.e. typically comparison implementation is done within the class.

Writable | Smart Insert | 55 : 5 : 1355

Search | 10:55 AM

You are screen sharing
Stop Share

To compare two objects, but not with its natural ordering
Use Comparator.
Typically Comparator provides comparison of two objects
outside that class.

```
// pre-defined Comparator<T> interface:
interface Comparator<T> {
    int compare(T obj1, T obj2);
}
```

Comparator is standard for comparing two given objects.
Returns difference between them.

```
   0 -- if obj1 == obj2
 +ve -- if obj1 > obj2
 -ve -- if obj1 < obj2
```

Writable
Smart Insert
55 : 5 : 1355
10:58 AM

day12 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Product1.java | Program02.java | Product2.java | Product3.java | Program03.java | Product.java

```
9                 new Product(3, "Pen", 45.0),
10                new Product(1, "Pencil", 5.0),
11                new Product(2, "Eraser", 3.0),
12                new Product(5, "Paper", 6.0),
13                new Product(4, "Notebook", 80.0)
14        };
15        System.out.println("Before Sort:");
16        for (int i = 0; i < arr.length; i++)
17            System.out.println(arr[i]);
18
19        class ProductNameComparator implements Comparator<Product> {
20            @Override
21            public int compare(Product x, Product y) {
22                int diff = x.getName().compareTo(y.getName());
23                return diff;
24            }
25        }
26
27        ProductNameComparator prodNameComparator = new ProductNameComparator();
28        Arrays.sort(arr, prodNameComparator);
29
30        System.out.println(" After Sort:");
31        for (int i = 0; i < arr.length; i++)
32            System.out.println(arr[i]);
33
34    }
```

Problems | Javadoc | Declaration | Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclips

```
Before Sort:
Product [id=3, name=Pen, price=45.0]
Product [id=1, name=Pencil, price=5.0]
Product [id=2, name=Eraser, price=3.0]
Product [id=5, name=Paper, price=6.0]
Product [id=4, name=Notebook, price=80.0]
 After Sort:
Product [id=2, name=Eraser, price=3.0]
Product [id=4, name=Notebook, price=80.0]
Product [id=5, name=Paper, price=6.0]
Product [id=3, name=Pen, price=45.0]
Product [id=1, name=Pencil, price=5.0]
```

Internally, whenever Arrays.sort() needs to compare array elems it will call comparator.compare() method.

Search 11:06 AM

day12 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
Collection<String> c = new LinkedList<>();
c.add("India");
c.add("Africa");
c.add("England");
c.add("USA");
c.add("India");
c.add("Australia");
c.add("West Indies");
System.out.println("Size: " + c.size()); // 7
System.out.println("toString(): " + c.toString());
// for-each loop
for(String ele : c)
    System.out.println(ele);
c.remove("USA");
System.out.println("toString(): " + c.toString()); // [India, Africa, England, India, Australia, West Indies]
c.remove("India");
System.out.println("toString(): " + c.toString()); // [Africa, England, India, Australia, West Indies]
// traverse the collection -- using Iterator
Iterator<String> trav = c.iterator();
while(trav.hasNext()) {
    String ele = trav.next();
    System.out.println(ele);
}
c.clear();
System.out.println("Size: " + c.size()); // 0
}
```

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot

```
toString(): [Africa, England, India, Australia, West Indies]
Africa
England
India
Australia
West Indies
Size: 0
```

ele = AF, ENG, IND, AUS, WI

Search 12:49 PM

for-each loop works for any class inherited from Iterable.

Core Java

Java Collection Framework

List interface

- Ordered/sequential collection.
- List can contain duplicate elements.
- List can contain multiple null elements.
- Elements can be accessed sequentially (bi-directional using Iterator) or randomly (index based).
- List enable searching (in the list)
- Implementations: ArrayList, Vector, Stack, LinkedList, etc.
- Abstract methods
  - void add(int index, E element)
  - E get(int index)
  - E set(int index, E element)
  - E remove(int index)
  - boolean addAll(int index, Collection<? extends E> c)
  - int indexOf(Object o)
  - int lastIndexOf(Object o)
  - String toString()
  - ListIterator listIterator()
  - ListIterator listIterator(int index)
  - List subList(int fromIndex, int toIndex)
- To store objects of user-defined types in the list, you must override equals() method for the objects. It is mandetory while searching operations like conatains(), indexOf(), lastIndexOf().

Iterator vs Enumeration

- Enumeration
  - Since Java 1.0
  - Methods
    - boolean hasMoreElements()
    - E nextElement()
  - Example

    ```
    Enumeration<E> e = v.elements();
    while(e.hasMoreElements()) {
        E ele = e.nextElement();
        System.out.println(ele);
    }
    ```

  - Enumeration behaves similar to fail-safe iterator.
- Iterator
  - Part of collection framework (1.2)

    - Methods
        - boolean hasNext()
        - E next()
        - void remove()
    - Example

```
Iterator<E> e = v.iterator();
while(e.hasNext()) {
    E ele = e.next();
    System.out.println(ele);
}
```

- ListIterator
    - Part of collection framework (1.2)
    - Inherited from Iterator
    - Bi-directional access
    - Methods
        - boolean hasNext()
        - E next()
        - int nextIndex()
        - boolean hasPrevious()
        - E previous()
        - int previousIndex()
        - void remove()
        - void set(E e)
        - void add(E e)

Traversal

- Using Iterator

```
Iterator<Integer> itr = list.iterator();
while(itr.hasNext()) {
    Integer i = itr.next();
    System.out.println(i);
}
```

- Using for-each loop

```
for(Integer i:list)
    System.out.println(i);
```

    - Gets converted into Iterator traversal

```
for(Iterator<Integer> itr = list.iterator(); itr.hasNext();)  {
    Integer i = itr.next();
    System.out.println(i);
}
```

- Enumeration -- Traversing Vector (Java 1.0)

```
// v is Vector<Integer>
Enumeration<Integer> e = v.elements();
while(e.hasMoreElements()) {
    Integer i = e.nextElement();
    System.out.println(i);
}
```

LinkedList class

- Internally LinkedList is doubly linked list.
- Elements can be traversed using Iterator, ListIterator, or using index.
- Primary use
  - Add/remove elements (anywhere)
  - Less contiguous memory available
- Limitations:
  - Slower random access
- Inherited from List<>, Deque<>.

ArrayList class

- Internally ArrayList is dynamic array (can grow or shrink dynamically).
- When ArrayList capacity is full, it grows by half of its size.
- Elements can be traversed using Iterator, ListIterator, or using index.
- Primary use
  - Random access
  - Add/remove elements (at the end)
- Limitations
  - Slower add/remove in between the collection
  - Uses more contiguous memory
- Inherited from List<>.

Vector class

- Internally Vector is dynamic array (can grow or shrink dynamically).
- Vector is a legacy collection (since Java 1.0) that is modified to fit List interface.
- Vector is synchronized (thread-safe) and hence slower.
- When Vector capacity is full, it doubles its size.
- Elements can be traversed using Enumeration, Iterator, ListIterator, or using index.
- Primary use

    - Random access
    - Add/remove elements (at the end)
- Limitations
    - Slower add/remove in between the collection
    - Uses more contiguous memory
    - Synchronization slow down performance in single threaded environment
- Inherited from List<>.

Fail-fast vs Fail-safe Iterator

- If state of collection is modified (add/remove operation other than iterator methods) while traversing a collection using iterator and iterator methods fails (with ConcurrentModificationException), then iterator is said to be Fail-fast.
    - e.g. Iterators from ArrayList, LinkedList, Vector, ...
- If iterator allows to modify the underlying collection (add/remove operation other than iterator methods) while traversing a collection (NO ConcurrentModificationException), then iterator is said to be Fail-safe.
    - e.g. Iterators from CopyOnWriteArrayList, ...

Synchronized vs Unsynchronized collections

- Synchronized collections are thread-safe and sync checks cause slower execution.
- Legacy collections were synchronized.
    - Vector
    - Stack
    - Hashtable
    - Properties
- Collection classes in collection framework (since 1.2) are non-synchronized (for better performance).
- Collection classes can be converted to synchronized collection using Collections class methods.
    - syncList = Collections.synchronizedList(list)
    - syncSet = Collections.synchronizedSet(set)
    - syncMap = Collections.synchronizedMap(map)

Collections class

- Helper/utility class that provides several static helper methods
- Methods
    - List reverse(List list);
    - List shuffle(List list);
    - void sort(List list, Comparator cmp)
    - E max(Collection list, Comparator cmp);
    - E min(Collection list, Comparator cmp);
    - List synchronizedList(List list);

Collection vs Collections

- Collection interface
    - All methods are public and abstract. They implemented in sub-classes.

- Since all methods are non-static, must be called on object.

```
Collection<Integer> list = new ArrayList<>();
//List<Integer> list = new ArrayList<>();
//ArrayList<Integer> list = new ArrayList<>();
list.remove(new Integer(12));
```

- Collections class
  - Helper class that contains all static methods.
  - We never create object of "Collections" class.

```
Collections.methodName(...);
```

Assignments

1. Store book details in a library in a list -- ArrayList.
   - Book details: isbn(string), price(double), authorName(string), quantity(int)
   - Write a menu driven program to
     1. Add new book in list
     2. Display all books in forward order
     3. Display all books in reverse order
     4. Delete a book at given index.

Vector vs ArrayList vs LinkedList

Vector v = new Vector();

* Dynamically growable/shrinkable array.
* Synchronized class / slower
* Legacy (1.0) * growth = 2 * capacity

Vector

ArrayList a = new ArrayList();

* Dynamically growable/shrinkable array.
* Non-synchronized / Faster
* Collection framework (1.2)
* growth = 1.5 * capacity

ArrayList

LinkedList l = new LinkedList();

* Doubly Linked List
* frequent add/delete op.
* slower random access.
* Inherited from List, Queue

To access elems one by one from any collection we use Iterator.

```
interface Iterator<T> {
    boolean hasNext();
    T next();
}
```

And to get Iterator obj of a collection we use Iterable.

```
interface Iterable<T>{
    Iterator<T> iterator();
}
```

CopyOnWriteArrayList

```
list = new CopyOnWriteArrayList();
Collections.addAll(list, 1,2,3,4,5);

itr1 = list.iterator();
while (itr1.hasNext()) {
    ele = itr1.next();
    sop(ele);
    if (ele == 3)
        itr1.add(6);
}
```

File Edit Selection View Go ···

You are screen sharing | Stop Share

day13.md | day12.md | classwork.md U | Untitled-1

day12.md > # Core Java > ## Day 12 - Agenda

```
# Core Java

## Day 12 - Agenda
* Generic Programming
    * Limitations
    * Generic Methods
    * Generic Interfaces
* Comparable
* Comparator
* Collection Framework
    * Collection

## Generic Programming

### Generic Methods
* Generic methods are used to implement generic algorithms.
* Example:
    ```Java
    // non type-safe
    void printArray(Object[] arr) {
        for(Object ele : arr)
            System.out.println(ele);
        System.out.println("Number of elements printed: " + arr.length);
    }
```

Natural Ordering

type-safety

int compareTo(T other);

"this" <=> other

```
class Emp implements Comparable<Emp> {
    // ...
    public int compareTo(Emp other) {
        int diff = this.id - other.id;
        return diff;
    }
}

main():
    Emp e1 = new Emp(...);
    Emp e2 = new Emp(...);
    int diff = e1.compareTo(e2);

    diff = e1.compareTo("James");
        // compiler error -- type-safety
```

main* | 0 0 | 0 | Ln 8, Col 13 | Spaces: 4 | UTF-8 | CRLF | Markdown

Search | 9:03 AM
```

File Edit Selection View Go ···

You are screen sharing | Stop Share

day13.md | day12.md | classwork.md U | Untitled-1

day12.md > # Core Java > ## Day 12 - Agenda

```
# Core Java

## Day 12 - Agenda
* Generic Programming
    * Limitations
    * Generic Methods
    * Generic Interfaces
* Comparable
* Comparator
* Collection Framework
    * Collection

## Generic Programming

### Generic Methods
* Generic methods are used to implement generic algorithms.
* Example:
    ```Java
    // non type-safe
    void printArray(Object[] arr) {
        for(Object ele : arr)
            System.out.println(ele);
        System.out.println("Number of elements printed: " + arr.length);
    }
```

int compare(T obj1, T obj2);

```
class Emp {
    // ....
}

class EmpComparator
    implements Comparator<Emp> {
    public int compare(Emp x, Emp y) {
        int diff = Double.compare(x.getSal(),
                                  y.getSal());
        return diff;
    }
}

main():
    Emp e1 = new Emp(...);
    Emp e2 = new Emp(...);
    EmpComparator cmp = new EmpComparator();
    int diff = cmp.compare(e1, e2);
```

main* | 0 0 | 0 | Ln 8, Col 13 | Spaces: 4 | UTF-8 | CRLF | Markdown

Search | 9:08 AM
```

File Edit Selection View Go ···

You are screen sharing | Stop Share

day13.md | day12.md | classwork.md U | Untitled-1

day12.md > # Core Java > ## Day 12 - Agenda

```md
# Core Java

## Day 12 - Agenda
* Generic Programming
    * Limitations
    * Generic Methods
    * Generic Interfaces
* Comparable
* Comparator
* Collection Framework
    * Collection

## Generic Programming

### Generic Methods
* Generic methods are used to implement generic algorithms.
* Example:
    ```Java
    // non type-safe
    void printArray(Object[] arr) {
        for(Object ele : arr)
            System.out.println(ele);
        System.out.println("Number of elements printed: " + arr.length);
    }
```

Emp[] arr = new Emp[] { .... };

Arrays.sort(arr); ← Internally use Comparable/Natural ordering for sorting/comparing objects. The class MUST BE inherited from Comparable interface.

Arrays.sort(arr, new EmpComparator()); ← Internally use Comparator (given in arg2) to sort/compare Emp array.

main* | 0 0 | 0 | Ln 8, Col 13 | Spaces: 4 | UTF-8 | CRLF | Markdown

Search | 9:12 AM
```

day13 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java

```
        // bi-directional traversal
    }
}
```

Iterator — hasNext(), next()

ListIterator — hasNext(), next(), hasPrevious(), previous()

Writable | Smart Insert | 85 : 1 : 2042

Search | 10:23 AM

day13 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java

```java
        // itr is by default refers to first element
        ListIterator<String> trav = l.listIterator();
        while(trav.hasNext()) {
            String ele = trav.next();
            System.out.print(ele + ", ");
        }
        System.out.println();

        // bi-directional traversal -- reverse direction
        System.out.print("REV Order: ");
        // itr should refer to after the last element
        trav = l.listIterator(l.size());
        while(trav.hasPrevious()) {
            String ele = trav.previous();
            System.out.print(ele + ", ");
        }
        System.out.println();
    }
}
```

check if cur ele is avail.

returns cur ele and takes trav to the next ele.

first

last

Ind Jpn Itl Eng Usa Aus Grm

trav (init pos)

first(0)

last(6) size(7)

Ind Jpn Itl Eng Usa Aus Grm

check if prev ele is avail.

trav (init pos)

takes trav to the prev elem and then return that elem.

Problems Javadoc Declaration Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 14, 2024, 10:31:16 AM – 10:31:17 AM) [pid: 9032]

```
lastIndexOf() - Element Found: USA at index: 4
FWD Order: India, Japan, Italy, England, USA, Australia, Germany,
REV Order: Germany, Australia, USA, England, Italy, Japan, India,
```

Search 10:31 AM

day13 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Book.java

```java
public class Book {
    private int id;
    private String name;
    private String subject;
    private double price;

    @Override
    public boolean equals(Object o)
        if(o == null)
            return false;
        if(this == o)
            return true;
        if(!(o instanceof Book))
            return false;
        Book other = (Book) o;
        if(this.id == other.id)
            return true;
        return false;
```

Program02.java

```java
        while(trav.hasPrevious()) {
            Book ele = trav.previous();
            System.out.println(ele);
        }

        // searching -- find given element
        int id = 5; // sc.nextInt();
        Book key = new Book();
        key.setId(id);
        int idx = list.indexOf(key);
        if(idx != -1) {
            Book ele = list.get(idx);
            System.out.println("Found at Index: " + idx + " " + ele);
        }
        else
            System.out.println("Book Not Found.");
    }
}
```

ArrayList class has implemented indexOf method:

```java
int indexOf(Object key) {
    for(int i=0; i<size(); i++) {
        T ele = get(i);
        if(key.equals(ele))
            return true;
    }
    return false;
}
```

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 14, 2024, 11:02:47 AM – 11:02:47 AM) [pid: 5744]

```
Book [id=5, name=The Fountainhead, subject=Novel, price=652.73]
Book [id=1, name=The Archer, subject=Novel, price=723.53]
Book [id=4, name=The Alchemist, subject=Novel, price=493.23]
Found at Index: 2 Book [id=5, name=The Fountainhead, subject=Novel, price=652.73]
```

Search 11:03 AM

day13 - demo02/src/com/sunbeam/Book.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java | Program01.java | Book.java

```java
public class Book implements Comparable<Book> {
    private int id;
    private String name;
    private String subject;
    private double price;

    @Override
    public boolean equals(Object o) {
        if(o == null)
            return false;
        if(this == o)
            return true;
        if(!(o instanceof Book))
            return false;
        Book other = (Book) o;
        if(this.id == other.id)
            return true;
        return false;
    }

    @Override
    public int compareTo(Book o) {
        int diff = this.id - o.id;
        return diff;
    }
```

It is recommended to have Comparable implementation compatible with equals() implementation of that class.

i.e. if two objects are equal by equals() method, then compareTo() should return 0 difference.

In other words, both methods should compare on the same fields of the class.

Writable | Smart Insert | 29 : 1 : 521

Search 11:39 AM

day13 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java × Program01.java Book.java

```java
        while(trav.hasPrevious()) {
            Book ele = trav.previous();
            System.out.println(ele);
        }

        // searching -- find given element -- index
        //  to function these methods correctly, th
        //  and equals() should compare on desired
        int id = 5; // sc.nextInt();
        Book key = new Book();
        key.setId(id);
        int idx = list.indexOf(key);
        if(idx != -1) {
            Book ele = list.get(idx);
            System.out.println("Found at Index: " + idx + " " + ele.toString());
        }
        else
            System.out.println("Book Not Found.");

        Collections.sort(list); // natural ordering -- Comparable in Book
        System.out.println("Asc Sorted Books (using Comparable)");
        for (Book bk : list)
            System.out.println(bk);
    }
}
```

Problems Javadoc Declaration Console ×

```
<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win3:
Found at Index: 2 Book [id=5, name=The Fountainhead, subject=Novel, |
Asc Sorted Books (using Comparable)
Book [id=1, name=The Archer, subject=Novel, price=723.53]
Book [id=2, name=Atlas Shrugged, subject=Novel, price=872.94]
Book [id=3, name=Lord of Rings, subject=Novel, price=621.53]
Book [id=4, name=The Alchemist, subject=Novel, price=493.23]
Book [id=5, name=The Fountainhead, subject=Novel, price=652.73]
Book [id=6, name=Harry Potter, subject=Novel, price=423.68]
```

Search 11:42 AM

day13 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java | Program01.java | Book.java

```java
        Book key = new Book();
        key.setId(id);
        int idx = list.indexOf(key);
        if(idx != -1) {
            Book ele = list.get(idx);
            System.out.println("Found at Index: " +
        }
        else
            System.out.println("Book Not Found.");

        Collections.sort(list); // natural ordering
        System.out.println("Asc Sorted Books (using Comparable -- Collections.sort())");
        for (Book bk : list)
            System.out.println(bk);

        System.out.println("Asc Sorted Books (using Comparator -- Collections.sort())");
        class BookNameComparator implements Comparator<Book> {
            @Override
            public int compare(Book x, Book y) {
                int diff = x.getName().compareTo(y.getName());
                return diff;
            }
        }
        Collections.sort(list, new BookNameComparator());
        for (Book bk : list)
            System.out.println(bk);
```

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32

```
Book [id=6, name=Harry Potter, subject=Novel, price=423.68]
Asc Sorted Books (using Comparator -- Collections.sort())
Book [id=2, name=Atlas Shrugged, subject=Novel, price=872.94]
Book [id=6, name=Harry Potter, subject=Novel, price=423.68]
Book [id=3, name=Lord of Rings, subject=Novel, price=621.53]
Book [id=4, name=The Alchemist, subject=Novel, price=493.23]
Book [id=1, name=The Archer, subject=Novel, price=723.53]
Book [id=5, name=The Fountainhead, subject=Novel, price=652.73]
```

Search 11:45 AM

day13 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java × Program01.java Book.java

```
53
54          System.out.println("Asc Sorted Books (using
55          class BookNameComparator implements Compara
56              @Override
57              public int compare(Book x, Book y) {
58                  int diff = x.getName().compareTo(y.
59                  return diff;
60              }
61          }
62          Collections.sort(list, new BookNameComparat
63          for (Book bk : list)
64              System.out.println(bk);
65
66          System.out.println("Asc Sorted Books (using Comparator -- list.sort())");
67          class BookPriceComparator implements Comparator<Book> {
68              @Override
69              public int compare(Book x, Book y) {
70                  int diff = Double.compare(x.getPrice(), y.getPrice());
71                  return diff;
72              }
73          }
74          list.sort(new BookPriceComparator());
75          for (Book bk : list)
76              System.out.println(bk);
77
78      }
```

Problems Javadoc Declaration Console ×

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32

```
Book [id=5, name=The Fountainhead, subject=Novel, price=652.73]
Asc Sorted Books (using Comparator -- list.sort())
Book [id=6, name=Harry Potter, subject=Novel, price=423.68]
Book [id=4, name=The Alchemist, subject=Novel, price=493.23]
Book [id=3, name=Lord of Rings, subject=Novel, price=621.53]
Book [id=5, name=The Fountainhead, subject=Novel, price=652.73]
Book [id=1, name=The Archer, subject=Novel, price=723.53]
Book [id=2, name=Atlas Shrugged, subject=Novel, price=872.94]
```

Search 11:47 AM

day13 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java

```java
package com.sunbeam;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

public class Program03 {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        Collections.addAll(list, 11, 22, 33, 44, 55);

        Iterator<Integer> itr = list.iterator();
        while(itr.hasNext()) {
            int ele = itr.next();
            System.out.println(ele);
            if(ele == 33)
                list.add(4, 66);
        }
    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x
11
22
33
Exception in thread "main" java.util.ConcurrentModificationException
        at java.base/java.util.ArrayList$Itr.checkForComodification(Ar
        at java.base/java.util.ArrayList$Itr.next(ArrayList.java:967)
        at com.sunbeam.Program03.main(Program03.java:15)
```

When an iterator is accessing elements one-by-one and meanwhile the collection is structurally modified, then iterator methods (like next()) will throw the ConcurrentModificationException.

Such Iterator is called as "Fail-fast Iterator".

Search 11:56 AM

day13 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java

```
                list.add(4, 66);
            }
        }
        */

        public static void main(String[] args) {
            List<Integer> list = new CopyOnWriteArrayList<>();
            Collections.addAll(list, 11, 22, 33, 44, 55);
            // fail-safe iterator
            Iterator<Integer> itr = list.iterator();
            while(itr.hasNext()) {
                int ele = itr.next();
                System.out.println(ele);
                if(ele == 33)
                    list.add(4, 66);
            }

            System.out.println("\n Updated List: ");
            itr = list.iterator();
            while(itr.hasNext()) {
                int ele = itr.next();
                System.out.println(ele);
            }
        }
    }

```

<terminated> Program03 [Java Applic

```
11
22
33
44
55

 Updated List:
11
22
33
44
66
55
```

Note that, the changes in collection may not be seen with current iterator. But it will be definately seen in new iterator.

When iterator is accessing elements one-by-one and meanwhile collection is structurally modified, the iterator methods will not throw any exception. Such iterators are "Fail-Safe Iterators".

The collections from java.util package have fail-fast iterators e.g. ArrayList, LinkedList, HashSet, ArrayDeque

The collections from java.util.concurrent package have fail-safe iterators e.g. CopyOnWriteArrayList, ...

Search 12:05 PM

day13 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java

```
                list.add(4, 66);
            }
        }
        */

        public static void main(String[] args) {
            List<Integer> list = new CopyOnWriteArrayList<>();
            Collections.addAll(list, 11, 22, 33, 44, 55);
            // fail-safe iterator
            Iterator<Integer> itr = list.iterator();
            while(itr.hasNext()) {
                int ele = itr.next();
                System.out.println(ele);
                if(ele == 33)
                    list.add(4, 66);
            }

            System.out.println("\n Updated List: ");
            itr = list.iterator();
            while(itr.hasNext()) {
                int ele = itr.next();
                System.out.println(ele);
            }
        }
    }

```

<terminated> Program03 [Java Applic

```
11
22
33
44
55

 Updated List:
11
22
33
44
66
55
```

Note that, the changes in collection may not be seen with current iterator. But it will be definately seen in new iterator.

When iterator is accessing elements one-by-one and meanwhile collection is structurally modified, the iterator methods will not throw any exception. Such iterators are "Fail-Safe Iterators".

The collections from java.util package have fail-fast iterators e.g. ArrayList, LinkedList, HashSet, ArrayDeque

The collections from java.util.concurrent package have fail-safe iterators e.g. CopyOnWriteArrayList, ...

Search 12:05 PM

day13 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
        // works for all "Collection".
        System.out.println("Traversal using Iterator: ");
        Iterator<Double> itr = list.iterator();
        while(itr.hasNext()) {
            Double ele = itr.next();
            System.out.print(ele + ", ");
        }
        System.out.println("\n");

        // works for all "List".
        System.out.println("Traversal using for loop - with index: ");
        for(int i = 0; i < list.size(); i++) {
            Double ele = list.get(i);
            System.out.print(ele + ", ");
        }
        System.out.println("\n");

        // works for "Vector".
        System.out.println("Traversal using Enumeration (Vector): ");
        Enumeration<Double> en = list.elements();
        while(en.hasMoreElements()) {
            Double ele = en.nextElement();
            System.out.print(ele + ", ");
        }
    }
```

Enumeration:
- hasMoreElements()
- nextElement()

Iterator:
- hasNext()
- next()
- remove() -- removes current elem

ListIterator:
- hasNext(), next()
- hasPrevious(), previous()
- nextIndex(), previousIndex()
- remove(), add(), set()

Writable | Smart Insert | 38 : 34 : 1141

Search | 12:24 PM

Java collection framework

Collection: basic collection i.e. add ele, remove ele, traverse collection, etc.

List: allow duplicate, random access, bi-direction traversal, searching.

Set : allow unique elems only

Queue : utility ds like stack & queue

LIFO - push(), pop(), peek(),

Map: key value pairs for fast searching.

day13 - demo06/src/com/sunbeam/Program06.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java | Program04.java | Program06.java

```java
package com.sunbeam;

import java.util.Stack;

public class Program06 {
    public static void main(String[] args) {
        Stack<Integer> s = new Stack<>();
        s.push(11);
        s.push(22);
        s.push(33);
        s.push(44);
        s.push(55);

        Integer ele = s.peek(); // 55
        System.out.println("Topmost Ele: " + ele);

        while(!s.isEmpty()) {
            Integer num = s.pop();
            System.out.println("Popped: " + num);
        }
    }
}
```

Stack is "Last In First Out"

Problems | Javadoc | Declaration | Console

<terminated> Program06 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\

```
Topmost Ele: 55
Popped: 55
Popped: 44
Popped: 33
Popped: 22
Popped: 11
```

Search 1:02 PM

Core Java

Collection framework

Queue interface

- Represents utility data structures (like Stack, Queue, ...) data structure.
- Implementations: LinkedList, ArrayDeque, PriorityQueue.
- Can be accessed using iterator, but no random access.
- Methods
    - boolean add(E e) - throw IllegalStateException if full.
    - E remove() - throw NoSuchElementException if empty
    - E element() - throw NoSuchElementException if empty
    - boolean offer(E e) - return false if full.
    - E poll() - returns null if empty
    - E peek() - returns null if empty
- In queue, addition and deletion is done from the different ends (rear and front).

Deque interface

- Represents double ended queue data structure i.e. add/delete can be done from both the ends.
- Two sets of methods
    - Throwing exception on failure: addFirst(), addLast(), removeFirst(), removeLast(), getFirst(), getLast().
    - Returning special value on failure: offerFirst(), offerLast(), pollFirst(), pollLast(), peekFirst(), peekLast().
- Can used as Queue as well as Stack.
- Methods
    - boolean offerFirst(E e)
    - E pollFirst()
    - E peekFirst()
    - boolean offerLast(E e)
    - E pollLast()
    - E peekLast()

ArrayDeque class

- Internally ArrayDeque is dynamically growable array.
- Elements are allocated contiguously in memory.

LinkedList class

- Internally LinkedList is doubly linked list.

PriorityQueue class

- Internally PriorityQueue is a "binary heap" data structure.
- Elements with highest priority is deleted first (NOT FIFO).
- Elements should have natural ordering or need to provide comparator.

Set interface

- Collection of unique elements (NO duplicates allowed).
- Implementations: HashSet, LinkedHashSet, TreeSet.
- Elements can be accessed using an Iterator.
- Abstract methods (same as Collection interface)
    - add() returns false if element is duplicate

HashSet class

- Non-ordered set (elements stored in any order)
- Elements must implement equals() and hashCode()
- Fast execution

LinkedHashSet class

- Ordered set (preserves order of insertion)
- Elements must implement equals() and hashCode()
- Slower than HashSet

SortedSet interface

- Use natural ordering or Comparator to keep elemenrs in sorted order
- Methods
    - E first()
    - E last()
    - SortedSet headSet(E toElement)
    - SortedSet subSet(E fromElement, E toElement)
    - SortedSet tailSet(E fromElement)

NavigableSet interface

- Sorted set with additional methods for navigation
- Methods
    - E higher(E e)
    - E lower(E e)
    - E pollFirst()
    - E pollLast()
    - NavigableSet descendingSet()
    - Iterator descendingIterator()

TreeSet class

- Sorted navigable set (stores elements in sorted order)

- Elements must implement Comparable or provide Comparator
- Slower than HashSet and LinkedHashSet
- It is recommended to have consistent implementation for Comparable (Natural ordering) and equals() method i.e. equality and comparison should done on same fields.
- If need to sort on other fields, use Comparator.

```
class Book implememts Comparable<Book> {
    private String isbn;
    private String name;
    // ...
    public int hashCode() {
        return isbn.hashCode();
    }
    public boolean equals(Object obj) {
        if(!(obj instanceof Book))
            return false;
        Book other=(Book)obj;
        if(this.isbn.equals(other.isbn))
            return true;
        return false;
    }
    public int compareTo(Book other) {
        return this.isbn.compareTo(other.isbn);
    }
}
```

```
// Store in sorted order by name
set = new TreeSet<Book>((b1,b2) -> b1.getName().compareTo(b2.getName()));
```

```
// Store in sorted order by isbn (Natural ordering)
set = new TreeSet<Book>();
```

HashTable Data structure

- Hashtable stores data in key-value pairs so that for the given key, value can be searched in fastest possible time.
- Internally hash-table is a table(array), in which each slot(index) has a bucket(collection). Key-value entries are stored in the buckets depending on hash code of the "key".
- Load factor = Number of entries / Number of buckets.
- Examples
  - Key=pincode, Value=city/area
  - Key=Employee, Value=Manager
  - Key=Department, Value=list of Employees

hashCode() method

- Object class has hashCode() method, that returns a unique number for each object (by converting its address into a number).
- To use any hash-based data structure hashCode() and equals() method must be implemented.
- If two distinct objects yield same hashCode(), it is referred as collision. More collisions reduce performance.
- Most common technique is to multiply field values with prime numbers to get uniform distribution and lesser collsions.
- hashCode() overriding rules
    - hash code should be calculated on the fields that decides equality of the object.
    - hashCode() should return same hash code each time unless object state is modified.
    - If two objects are equal (by equals()), then their hash code must be same.
    - If two objects are not equal (by equals()), then their hash code may be same (but reduce performance).

Map interface

- Collection of key-value entries (Duplicate "keys" not allowed).
- Implementations: HashMap, LinkedHashMap, TreeMap, Hashtable, ...
- The data can be accessed as set of keys, collection of values, and/or set of key-value entries.
- Map.Entry<K,V> is nested interface of Map<K,V>.
    - K getKey()
    - V getValue()
    - V setValue(V value)
- Abstract methods

```
* boolean isEmpty()
* int size()
* V put(K key, V value)
* V get(Object key)
* Set<K> keySet()
* Collection<V> values()
* Set<Map.Entry<K,V>> entrySet()
* boolean containsValue(Object value)
* boolean containsKey(Object key)
* V remove(Object key)
* void clear()
* void putAll(Map<? extends K,? extends V> map)
```

- Maps not considered as true collection, because it is not inherited from Collection interface.

HashMap class

- Non-ordered map (entries stored in any order -- as per hash code of key)
- Keys must implement equals() and hashCode()
- Fast execution
- Mostly used Map implementation

LinkedHashMap class

- Ordered map (preserves order of insertion)
- Keys must implement equals() and hashCode()
- Slower than HashSet
- Since Java 1.4

TreeMap class

- Sorted navigable map (stores entries in sorted order of key)
- Keys must implement Comparable or provide Comparator
- Slower than HashMap and LinkedHashMap
- Internally based on Red-Black tree.
- Doesn't allow null key (allows null value though).

Hashtable class

- Similar to HashMap class.
- Legacy collection class (since Java 1.0), modified for collection framework (Map interface).
- Synchronized collection -- Thread safe but slower performance
- Inherited from java.util.Dictionary abstract class (it is Obsolete).

Queue

Queue D.S.	Stack D.S.
① FIFO ② Add/Remove is from Diff ends (Rear/Front)	① LIFO ② Add/Remove is from Same end (top).

Hash Table Data Structure

Hashtable →

- very fast search
- key-value
- ideal search: O(1)

Example:

- Name → Mobile
- Roll → Student
- pin → City/Area

key value

key $\xrightarrow[\text{fn}]{\text{hash}}$ index of array

$h(k) = k \% size$

$= k \% 10$

Hash Table Data Structure

Hashtable →

- very fast search
- key-value
- ideal search: O(1)

Example:

- Name → Mobile
- Roll → Student
- [illegible]

key value

key $\xrightarrow[\text{fn}]{\text{hash}}$ index of array

$h(k) = k \% size$

$= k \% 10$

Hash Table — Collision

key $\xrightarrow[\text{fn}]{\text{hash}}$ index of array / slot of table OR

$h(k) = k \% size$

$= k \% 10$

different keys corresponds to the same slot in the table → collision.

Collision handling

- Open addressing → if Load Factor <= 1.0
- separate chaining → irrespective of Load Factor

Hash Tables

Load Factor

$$= \frac{\text{Number of entries}}{\text{Number of slots}}$$

Case 1: Entries < Slots

e.g. Load Factor $= \frac{7}{10}$

i.e. 0.7 (< 1)

Case 2: Entries = Slots

e.g. Load Factor $= \frac{10}{10}$

i.e. 1.0 (= 1)

Case 3: Entries > Slots

e.g. Load Factor $= \frac{12}{10}$

i.e. 1.2 (> 1)

Hash Table — Separate Chaining

Each slot in table holds a collection of entries - called as buckets.

when collision occurs, the new entry will be added into corresponding bucket.

key $\xrightarrow[\text{fn}]{\text{hash}}$ index of array / slot of table OR

$h(k) = k \% size$

$= k \% 10$

$$\text{Load factor} = \frac{\text{Num of entries}}{\text{Num of buckets}}$$

Java — Hash tables

Java has built-in hash table implementations.

1. HashMap
2. LinkedHashMap
3. TreeMap
4. Hashtable (legacy)
5. Properties (legacy)

Programmer should calculate hash value of the key - override hashCode() method.

The slot in the table is calculated internally by → slot = key.hashCode() % size;

1. equal objects must yield same hashCode.
2. un-equal objects should ideally yield different hashCode.
   if hash code of unequal objects is same, it will cause collision.
3. hash code of an object must be consistent i.e. hashCode() should return same value unless object state is modified.

Overriding hashCode()

```
class Distance {
    int feet, inches;
    ⋮
    equals()
        → feet
        → inches
    @Override
    public int hashCode() {
        int hash = feet + inches;
        return hash;
    }
}
```

d1 → 5' 8" → hash code = 13
d2 → 5' 8" → hash code = 13
d3 → 8' 5" → hash code = 13

```
class Distance {
    int feet, inches;
    ⋮
    equals()
        → feet
        → inches
    @Override
    public int hashCode() {
        int hash = 31*feet + inches;
        return hash;
    }
}
```

d1 → 5' 8" → hash code = 168
d2 → 5' 8" → hash code = 168
d3 → 8' 5" → hash code = 253

```
class Distance {
    int feet, inches;
    ⋮
    equals()
        → feet
        → inches
    @Override
    public int hashCode() {
        int hash =
            Objects.hash(feet, inches);
        return hash;
    }
}
```

Sets and Maps

HashSet<K> = HashMap<K, null>

LinkedHashSet<K> = LinkedHashMap<K, null>

} duplication based on equals() + hashCode() of "K".

TreeSet<K> = TreeMap<K, null>

} duplication based on Comparable of "K" or Comparator of "K" given in Constructor.

Duplicate elems not allowed.

Duplicate keys not allowed.

Java collection framework

Collection: basic collection i.e. add ele, remove ele, traverse collection, etc.

List: allow duplicate, random access, bi-direction traversal, searching.

Set: allow unique elems only

Queue: utility ds like stack & queue

Map: key value pairs for fast searching.

day14 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.8]
  - src
    - com.sunbeam
      - Program01.java

Program01.java

```java
import java.util.Queue;

public class Program01 {
    public static void main(String[] args) {
        Queue<String> q = new LinkedList<>();
        q.offer("One");
        q.offer("Two");
        q.offer("Three");
        q.offer("Four");
        System.out.println("First Element: " + q.peek());
        while(!q.isEmpty()) {
            String ele = q.poll();
            System.out.println("Popped: "  +ele);
        }
        System.out.println("Popped from Empty Queue: " + q.poll()); // null
    }
}
```

if operation fails, it returns null.

return null, because queue is empty.

Problems | Javadoc | Declaration | Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 202

```
First Element: One
Popped: One
Popped: Two
Popped: Three
Popped: Four
Popped from Empty Queue: null
```

Writable | Smart Insert | 18 : 76 : 491

Search | 9:26 AM

day14 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.8]
  - src
    - com.sunbeam
      - Program01.java

Program01.java

```java
    public static void main(String[] args) {
        //Queue<String> q = new LinkedList<>();
        Queue<String> q = new ArrayDeque<>();
        q.add("One");
        q.add("Two");
        q.add("Three");
        q.add("Four");
        System.out.println("First Element: " + q.element());
        while(!q.isEmpty()) {
            String ele = q.remove();
            System.out.println("Popped: "  +ele);
        }
        System.out.println("Popped from Empty Queue: " + q.remove()); // null
    }
}
```

If any operation fails, it throws exception.

throws NoSuchelementException because queue is empty.

Problems | Javadoc | Declaration | Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 20

```
First Element: One
Popped: One
Popped: Two
Popped: Three
Popped: Four
Exception in thread "main" java.util.NoSuchElementException
        at java.base/java.util.ArrayDeque.removeFirst(ArrayDeque.java:362)
```

Syntax error on token "/", delete this token | Writable | Smart Insert | 37 : 9 : 940

Search | 9:28 AM

day14 - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.8]
  - src
    - com.sunbeam
      - Program01.java

Deque as a Stack
==========
offerFirst()
pollFirst()
peek()

Last In First Out

Deque as a Queue
==========
offerLast()
pollFirst()
peek()

First In First Out

Problems Javadoc Declaration Console

<terminated> Program01 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 20

```
First Element: One
Popped: One
Popped: Two
Popped: Three
Popped: Four
Exception in thread "main" java.util.NoSuchElementException
        at java.base/java.util.ArrayDeque.removeFirst(ArrayDeque.java:362)
```

Search 9:30 AM

day14 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
import java.util.LinkedList;

public class Program02 {
    public static void main(String[] args) {
        // Deque as Stack
        Deque<Integer> s = new ArrayDeque<>(); //new LinkedList<>();
        s.offerFirst(11);
        s.offerFirst(22);
        s.offerFirst(33);
        s.offerFirst(44);
        while(!s.isEmpty()) {
            Integer ele = s.pollFirst();
            System.out.println("Popped: " + ele);
        }
    }
}
```

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 9:43:03 AM – 9:43:03 AM) [pid: 18556]

```
Popped: 44
Popped: 33
Popped: 22
Popped: 11
```

The word 'Deque' is not correctly spelled | Writable | Smart Insert | 9 : 26 [17]

Search 9:43 AM

day14 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
        }
    }
    */

    public static void main(String[] args) {
        // Deque as Queue
        Deque<Integer> s = new ArrayDeque<>(); //new LinkedList<>();
        s.offerLast(11);
        s.offerLast(22);
        s.offerLast(33);
        s.offerLast(44);
        while(!s.isEmpty()) {
            Integer ele = s.pollFirst();
            System.out.println("Popped: " + ele);
        }
    }
```

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 9:43:43 AM – 9:43:43 AM) [pid: 4064]

```
Popped: 11
Popped: 22
Popped: 33
Popped: 44
```

The word 'Deque' is not correctly spelled | Writable | Smart Insert | 24 : 26 [17]

Search 9:43 AM

day14 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.8]
  - src
    - com.sunbeam
      - Program01.java
- demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.8]
  - src
    - com.sunbeam
      - Program02.java
- demo03 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.8]
  - src
    - com.sunbeam
      - Program03.java

Program03.java

```java

public class Program03 {
    public static void main(String[] args) {
        // Elements are retrieved as per priority -- decided by Comparable (Natural Ordering)
        Queue<String> q = new PriorityQueue<>();
        q.offer("I");
        q.offer("N");
        q.offer("F");
        q.offer("O");
        q.offer("T");
        q.offer("E");
        q.offer("C");
        q.offer("H");
        while(!q.isEmpty()) {
            String ele = q.poll();
            System.out.print(ele + ", ");
        }
    }
```

Problems Javadoc Declaration Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 20

```
C, E, F, H, I, N, O, T,
```

Writable | Smart Insert | 8 : 94 [85]

Search 9:48 AM

day14 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java

```java
    public static void main(String[] args) {
        class StringDescComparator implements Comparator<String> {
            @Override
            public int compare(String x, String y) {
                return - x.compareTo(y);
            }
        }
        // Elements are retrieved as per priority -- decided by given Comparator.
        Queue<String> q = new PriorityQueue<>(new StringDescComparator());
        q.offer("T");
        q.offer("E");
        q.offer("C");
        q.offer("H");
        while(!q.isEmpty()) {
            String ele = q.poll();
            System.out.print(ele + ", ");
        }
    }
```

Problems Javadoc Declaration Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 9:51:29 AM – 9:51:29 AM) [pid: 12816]

```
T, H, E, C,
```

Writable | Smart Insert | 34 : 73 [26]

Search 9:51 AM

day14 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java

```java
    public static void main(String[] args) {
        class StringDescComparator implements Comparator<String> {
            @Override
            public int compare(String x, String y) {
                return - x.compareTo(y);
            }
        }
        // Elements are retrieved as per priority -- decided by given Comparator.
        Queue<String> q = new PriorityQueue<>(new StringDescComparator());
        q.offer("T");
        q.offer("E");
        q.offer("C");
        q.offer("H");
        while(!q.isEmpty()) {
            String ele = q.poll();
            System.out.print(ele + ", ");
        }
    }
```

offer()/poll() -- time complexity = O(log n)

Internally uses Heap data structure.

Problems Javadoc Declaration Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 9:51:29 AM – 9:51:29 AM) [pid: 12816]

```
T, H, E, C,
```

Writable | Smart Insert | 38 : 22 : 975

Search 9:53 AM

day14 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
    public static void main(String[] args) {
        // Deque as Stack
        Deque<Integer> s = new ArrayDeque<>(); //new LinkedList<>();
        s.offerFirst(11);
        s.offerFirst(22);
        s.offerFirst(33);
        s.offerFirst(44);
        while(!s.isEmpty()) {
            Integer ele = s.pollFirst();
            System.out.println("Popped: " + ele);
        }
    }
    */

    public static void main(String[] args) {
        // Deque as Queue
        Deque<Integer> s = new ArrayDeque<>(); //new LinkedList<>();
        s.offerLast(11);
        s.offerLast(22);
        s.offerLast(33);
        s.offerLast(44);
        while(!s.isEmpty()) {
            Integer ele = s.pollFirst();
            System.out.println("Popped: " + ele);
        }
```

Time Complexity:

add element -- O(1)

remove element -- O(1)

Writable | Smart Insert | 1 : 1 : 0

Search 9:54 AM

day14 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.TreeSet;

public class Program04 {
    public static void main(String[] args) {
        //Set<String> set = new HashSet<>();
        //Set<String> set = new LinkedHashSet<>();
        Set<String> set = new TreeSet<>();

        set.add("India"); // return true
        set.add("Japan"); // return true
        set.add("India"); // return false - already exists
        set.add("Gemany"); // return true
        set.add("Africa"); // return true
        set.add("India"); // return false - already exists
        set.add("USA"); // return true
        set.add("Japan"); // return false - already exists

        System.out.println("Size: " + set.size());

        for (String str : set)
            System.out.println(str);
    }
}
```

Stores elements in sorted order (Natural order).

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.REL

```
Size: 5
Africa
Gemany
India
Japan
USA
```

Search 10:01 AM

day14 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.TreeSet;

public class Program04 {
    public static void main(String[] args) {
        //Set<String> set = new HashSet<>();
        Set<String> set = new LinkedHashSet<>();
        //Set<String> set = new TreeSet<>();

        set.add("India"); // return true
        set.add("Japan"); // return true
        set.add("India"); // return false - already exists
        set.add("Gemany"); // return true
        set.add("Africa"); // return true
        set.add("India"); // return false - already exists
        set.add("USA"); // return true
        set.add("Japan"); // return false - already exists

        System.out.println("Size: " + set.size());

        for (String str : set)
            System.out.println(str);
    }
}
```

Elements stored in order of Insertion.

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.REL

```
Size: 5
India
Japan
Gemany
Africa
USA
```

Search 10:02 AM

day14 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.TreeSet;

public class Program04 {
    public static void main(String[] args) {
        Set<String> set = new HashSet<>();
        //Set<String> set = new LinkedHashSet<>();
        //Set<String> set = new TreeSet<>();

        set.add("India"); // return true
        set.add("Japan"); // return true
        set.add("India"); // return false - already exists
        set.add("Gemany"); // return true
        set.add("Africa"); // return true
        set.add("India"); // return false - already exists
        set.add("USA"); // return true
        set.add("Japan"); // return false - already exists

        System.out.println("Size: " + set.size());

        for (String str : set)
            System.out.println(str);
    }
}
```

Elements are stored in arbitrary order - based on hashcode of each element.

Problems Javadoc Declaration Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.REL

```
Size: 5
USA
Japan
Gemany
Africa
India
```

Search 10:03 AM

day14 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java × Book.java

```java
package com.sunbeam;

import java.util.HashSet;
import java.util.Set;

public class Program05 {
    public static void main(String[] args) {
        Set<Book> set = new HashSet<>();
        set.add(new Book(4, "The Alchemist", "Novel", 493.23));
        set.add(new Book(1, "The Archer", "Novel", 723.53));
        set.add(new Book(5, "The Fountainhead", "Novel", 652.73));
        set.add(new Book(2, "Atlas Shrugged", "Novel", 872.94));
        set.add(new Book(6, "Harry Potter", "Novel", 423.68));
        set.add(new Book(1, "The Archer", "Novel", 723.53));
        set.add(new Book(3, "Lord of Rings", "Novel", 621.53));
        System.out.println("Set Size: " + set.size());
        for (Book b : set)
            System.out.println(b);
    }
}
```

Problems Javadoc Declaration Console ×

<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\or

```
Set Size: 7
Book [id=1, name=The Archer, subject=Novel, pr
Book [id=5, name=The Fountainhead, subject=Nov
Book [id=1, name=The Archer, subject=Novel, pr
Book [id=6, name=Harry Potter, subject=Novel,
Book [id=2, name=Atlas Shrugged, subject=Novel
Book [id=4, name=The Alchemist, subject=Novel,
Book [id=3, name=Lord of Rings, subject=Novel,
```

Elements are duplicated in HashSet, even if equals() is overridden in Book class. Because HashSet doesn't compare elements only on basis of equals().

NOTE: HashSet and LinkedHashSet considers elements equal if and only if their hashCode() is same AND calling equals() to compare them returns true.

Search 10:14 AM

day14 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java × Book.java

```java
package com.sunbeam;

import java.util.HashSet;
import java.util.Set;

public class Program05 {
    public static void main(String[] args) {
        Set<Book> set = new HashSet<>();
        set.add(new Book(4, "The Alchemist", "Novel", 493.23));
        set.add(new Book(1, "The Archer", "Novel", 723.53));
        set.add(new Book(5, "The Fountainhead", "Novel", 652.73));
        set.add(new Book(2, "Atlas Shrugged", "Novel", 872.94));
        set.add(new Book(6, "Harry Potter", "Novel", 423.68));
        set.add(new Book(1, "The Archer", "Novel", 723.53));
        set.add(new Book(3, "Lord of Rings", "Novel", 621.53));
        System.out.println("Set Size: " + set.size());
        for (Book b : set)
            System.out.println(b);
    }
}
```

Problems Javadoc Declaration Console ×

<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\or

```
Set Size: 6
Book [id=1, name=The Archer, subject=Novel, pr
Book [id=2, name=Atlas Shrugged, subject=Novel
Book [id=3, name=Lord of Rings, subject=Novel,
Book [id=4, name=The Alchemist, subject=Novel,
Book [id=5, name=The Fountainhead, subject=Nov
Book [id=6, name=Harry Potter, subject=Novel,
```

Duplicates are removed.

Book class implemented both hashCode() and equals(). -- implemented on "id" field.

Search 10:21 AM

day14 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java × Book.java

```java
    */

    public static void main(String[] args) {
        Set<Book> set = new TreeSet<>();

        set.add(new Book(4, "The Alchemist", "Novel", 493.23));
        set.add(new Book(1, "The Archer", "Novel", 723.53));
        set.add(new Book(5, "The Fountainhead", "Novel", 652.73));
        set.add(new Book(2, "Atlas Shrugged", "Novel", 872.94));
        set.add(new Book(6, "Harry Potter", "Novel", 423.68));
        set.add(new Book(1, "The Archer", "Novel", 723.53));
        set.add(new Book(3, "Lord of Rings", "Novel", 621.53));
        System.out.println("Set Size: " + set.size());
        for (Book b : set)
            System.out.println(b);
    }
}
```

To store elements in a TreeSet, they must be Comparable i.e. must have natural ordering.*

*NOTE: TreeSet param-less constructor is used.

Problems Javadoc Declaration Console ×

<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 10:25:26 AM – 10:25:26 AM) [pid: 1723

```
xception in thread "main" java.lang.ClassCastException: class com.sunbeam.Book cannot be cast to class java.lang.Comparable (
        at java.base/java.util.TreeMap.compare(TreeMap.java:1569)
        at java.base/java.util.TreeMap.addEntryToEmptyMap(TreeMap.java:776)
        at java.base/java.util.TreeMap.put(TreeMap.java:785)
        at java.base/java.util.TreeMap.put(TreeMap.java:534)
```

Search 10:25 AM

day14 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java × Book.java

```java
        set.add(new Book(2, "Atlas Shrugged", "Novel", 872.94));
        set.add(new Book(6, "Harry Potter", "Novel", 423.68));
        set.add(new Book(1, "The Archer", "Novel", 723.53));
        set.add(new Book(3, "Lord of Rings", "Novel", 621.53));
        System.out.println("Set Size: " + set.size());
        for (Book b : set)
            System.out.println(b);
    }
    */

    public static void main(String[] args) {
        Set<Book> set = new TreeSet<>();

        set.add(new Book(4, "The Alchemist", "Novel", 493.23));
        set.add(new Book(1, "The Archer", "Novel", 723.53));
        set.add(new Book(5, "The Fountainhead", "Novel", 652.73));
        set.add(new Book(2, "Atlas Shrugged", "Novel", 872.94));
        set.add(new Book(6, "Harry Potter", "Novel", 423.68));
        set.add(new Book(1, "The Archer", "Novel", 723.53));
        set.add(new Book(3, "Lord of Rings", "Novel", 621.53));
        System.out.println("Set Size: " + set.size());
        for (Book b : set)
            System.out.println(b);
    }
}
```

Problems Javadoc Declaration Console ×

<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclip

```
Set Size: 6
Book [id=1, name=The Archer, subject=Novel, price
Book [id=2, name=Atlas Shrugged, subject=Novel, p
Book [id=3, name=Lord of Rings, subject=Novel, pr
Book [id=4, name=The Alchemist, subject=Novel, pr
Book [id=5, name=The Fountainhead, subject=Novel,
Book [id=6, name=Harry Potter, subject=Novel, pri
```

Duplicates removed in TreeSet as per natural ordering of the elements.

The Book class implenented Comparable and compared on basis of "id".

Search 10:28 AM

day14 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java × Book.java

```java
class BookPriceComparator implements Comparator<Book> {
    @Override
    public int compare(Book x, Book y) {
        return Double.compare(x.getPrice(), y.getPrice());
    }
}
// stroes Books in sorted order -- as per Comparator (on price)
Set<Book> set = new TreeSet<>(new BookPriceComparator());
set.add(new Book(4, "The Alchemist", "Novel", 493.23));
set.add(new Book(1, "The Archer", "Novel", 723.53));
set.add(new Book(5, "The Fountainhead", "Novel", 652.73));
set.add(new Book(2, "Atlas Shrugged", "Novel", 872.94));
set.add(new Book(6, "Harry Potter", "Novel", 621.53));
set.add(new Book(1, "The Archer", "Novel", 723.53));
set.add(new Book(3, "Lord of Rings", "Novel", 621.53));
System.out.println("Set Size: " + set.size());
for (Book b : set)
```

Problems Javadoc Declaration Console ×

<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 10:34:15 AM – 10:34:16 AM) [pid: 1750

```
Set Size: 5
Book [id=4, name=The Alchemist, subject=Novel, price=493.23]
Book [id=6, name=Harry Potter, subject=Novel, price=621.53]
Book [id=5, name=The Fountainhead, subject=Novel, price=652.73]
Book [id=1, name=The Archer, subject=Novel, price=723.53]
Book [id=2, name=Atlas Shrugged, subject=Novel, price=872.94]
```

Stored in sorted order of price -- given by Comparator.

If price is same, element is removed (duplication).

Search 10:34 AM

day14 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java × Book.java

```java
class BookPriceComparator implements Comparator<Book> {
    @Override
    public int compare(Book x, Book y) {
        return Double.compare(x.getPrice(), y.getPrice());
    }
}
// stroes Books in sorted order -- as per Comparator (on price)
Set<Book> set = new TreeSet<>(new BookPriceComparator());
set.add(new Book(4, "The Alchemist", "Novel", 493.23));
set.add(new Book(1, "The Archer", "Novel", 723.53));
set.add(new Book(5, "The Fountainhead", "Novel", 652.73));
set.add(new Book(2, "Atlas Shrugged", "Novel", 872.94));
set.add(new Book(6, "Harry Potter", "Novel", 621.53));
set.add(new Book(1, "The Archer", "Novel", 723.53));
set.add(new Book(3, "Lord of Rings", "Novel", 621.53));
System.out.println("Set Size: " + set.size());
for (Book b : set)
```

Problems Javadoc Declaration Console ×

<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 10:34:15 AM – 10:34:16 AM) [pid: 1750

```
Set Size: 5
Book [id=4, name=The Alchemist, subject=Novel, price=493.23]
Book [id=6, name=Harry Potter, subject=Novel, price=621.53]
Book [id=5, name=The Fountainhead, subject=Novel, price=652.73]
Book [id=1, name=The Archer, subject=Novel, price=723.53]
Book [id=2, name=Atlas Shrugged, subject=Novel, price=872.94]
```

Stored in sorted order of price -- given by Comparator.

If price is same, element is removed (duplication).

Search 10:34 AM

String (Java Platform SE 8)
You are screen sharing
Stop Share
https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#hashCode--
133%

hashCode

```
public int hashCode()
```

Returns a hash code for this string. The hash code for a `String` object is computed as

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

using `int` arithmetic, where `s[i]` is the *i*th character of the string, `n` is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

Overrides:

`hashCode` in class `Object`

Returns:

a hash code value for this object.

See Also:

`Object.equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

indexOf

Search
12:25 PM

day14 - demo08/src/com/sunbeam/Program08.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program08.java

```java
public class Program08 {
    public static void main(String[] args) {
        Map<Integer, String> map = new HashMap<>();
        map.put(415110, "Karad - Satara"); // retruns -- null
        map.put(411052, "Hinjawadi - Pune"); // retruns -- null
        map.put(411046, "Katraj - Pune"); // retruns -- null
        map.put(400027, "Byculla - Mumbai"); // retruns -- null
        map.put(411002, "Bajirao Rd - Pune"); // retruns -- null
        map.put(411037, "Marketyard - Pune"); // retruns -- null
        map.put(411007, "Aundh - Pune"); // retruns -- null
        map.put(411052, "HINJEWADI - Pune"); // when key is duplicate, value is overwritten
            // returns -- old value for the key -- "Hinjawadi - Pune"
```

HashMap stores entries in arbitrary order.
Depends on hash code of key.

Problems Javadoc Declaration Console

<terminated> Program08 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 12:57:31 PM – 12:57:31 PM) [pid: 8520]

```
Keys (Pins): [415110, 411046, 411007, 411052, 411037, 411002, 400027]
Values (Areas): [Karad - Satara, Katraj - Pune, Aundh - Pune, HINJEWADI - Pune, Marketyard - Pune, Bajirao Rd - Pune, Byculla
415110 --> Karad - Satara
411046 --> Katraj - Pune
411007 --> Aundh - Pune
411052 --> HINJEWADI - Pune
411037 --> Marketyard - Pune
411002 --> Bajirao Rd - Pune
400027 --> Byculla - Mumbai
```

Writable | Smart Insert | 46 : 1 : 1658

Search 12:57 PM

day14 - demo08/src/com/sunbeam/Program08.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program08.java

```java
public class Program08 {
    public static void main(String[] args) {
        //Map<Integer, String> map = new HashMap<>();
        Map<Integer, String> map = new LinkedHashMap<>();
        map.put(415110, "Karad - Satara"); // retruns -- null
        map.put(411052, "Hinjawadi - Pune"); // retruns -- null
        map.put(411046, "Katraj - Pune"); // retruns -- null
        map.put(400027, "Byculla - Mumbai"); // retruns -- null
        map.put(411002, "Bajirao Rd - Pune"); // retruns -- null
        map.put(411037, "Marketyard - Pune"); // retruns -- null
        map.put(411007, "Aundh - Pune"); // retruns -- null
        map.put(411052, "HINJEWADI - Pune"); // when key is duplicate, value is overwritten
            // returns -- old value for the key -- "Hinjawadi - Pune"
```

LinkedHashMap stores elements in the order of insertion of keys.

Problems Javadoc Declaration Console

<terminated> Program08 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 12:58:50 PM – 12:58:51 PM) [pid: 13540

```
Keys (Pins): [415110, 411052, 411046, 400027, 411002, 411037, 411007]
Values (Areas): [Karad - Satara, HINJEWADI - Pune, Katraj - Pune, Byculla - Mumbai, Bajirao Rd - Pune, Marketyard - Pune, Aur
415110 --> Karad - Satara
411052 --> HINJEWADI - Pune
411046 --> Katraj - Pune
400027 --> Byculla - Mumbai
411002 --> Bajirao Rd - Pune
411037 --> Marketyard - Pune
411007 --> Aundh - Pune
```

Writable | Smart Insert | 14 : 58 : 387

Search 12:58 PM

day14 - demo08/src/com/sunbeam/Program08.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program08.java

```java
public class Program08 {
    public static void main(String[] args) {
        //Map<Integer, String> map = new HashMap<>();
        //Map<Integer, String> map = new LinkedHashMap<>();
        Map<Integer, String> map = new TreeMap<>();
        map.put(415110, "Karad - Satara"); // retruns -- null
        map.put(411052, "Hinjawadi - Pune"); // retruns -- null
        map.put(411046, "Katraj - Pune"); // retruns -- null
        map.put(400027, "Byculla - Mumbai"); // retruns -- null
        map.put(411002, "Bajirao Rd - Pune"); // retruns -- null
        map.put(411037, "Marketyard - Pune"); // retruns -- null
        map.put(411007, "Aundh - Pune"); // retruns -- null
        map.put(411052, "HINJEWADI - Pune"); // when key is duplicate, value is overwritten
```

TreeMap stores entries in natural ordering of keys (sorted).

NOTE: TreeMap() param less ctor is used.

Problems Javadoc Declaration Console

<terminated> Program08 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\javaw.exe (Feb 15, 2024, 1:00:04 PM – 1:00:04 PM) [pid: 21328]

```
Keys (Pins): [400027, 411002, 411007, 411037, 411046, 411052, 415110]
Values (Areas): [Byculla - Mumbai, Bajirao Rd - Pune, Aundh - Pune, Marketyard - Pune, Katraj - Pune, HINJEWADI - Pune, Karac
400027 --> Byculla - Mumbai
411002 --> Bajirao Rd - Pune
411007 --> Aundh - Pune
411037 --> Marketyard - Pune
411046 --> Katraj - Pune
411052 --> HINJEWADI - Pune
415110 --> Karad - Satara
```

Writable | Smart Insert | 16 : 52 : 463

Search 1:00 PM

Core Java

Annoymous Inner class

- Creates a new class inherited from the given class/interface and its object is created.
- If in static context, behaves like static member class. If in non-static context, behaves like non-static member class.
- Along with Outer class members, it can also access (effectively) final local variables of the enclosing method.

```
// (named) local class
class EmpnoComparator implements Comparator<Employee> {
    public int compare(Employee e1, Employee e2) {
        return e1.getEmpno() - e2.getEmpno();
    }
}
Arrays.sort(arr, new EmpnoComparator());    // anonymous obj of local class
```

```
// Anonymous inner class
Comparator<Employee> cmp = new Comparator<Employee>() {
    public int compare(Employee e1, Employee e2) {
        return e1.getEmpno() - e2.getEmpno();
    }
};
Arrays.sort(arr, cmp);
```

```
// Anonymous object of Anonymous inner class.
Arrays.sort(arr, new Comparator<Employee>() {
    public int compare(Employee e1, Employee e2) {
        return e1.getEmpno() - e2.getEmpno();
    }
});
```

Java 8 Interfaces

- Before Java 8
  - Interfaces are used to design specification/standards. It contains only declarations – public abstract.

```
interface Geometry {
    /*public static final*/ double PI = 3.14;
```

```
    /*public abstract*/ int calcRectArea(int length, int breadth);
    /*public abstract*/ int calcRectPeri(int length, int breadth);
}
```

  - As interfaces doesn't contain method implementations, multiple interface inheritance is supported (no ambiguity error).
  - Interfaces are immutable. One should not modify interface once published.
- Java 8 added many new features in interfaces in order to support functional programming in Java. Many of these features also contradicts earlier Java/OOP concepts.

Default methods

- Java 8 allows default methods in interfaces. If method is not overridden, its default implementation in interface is considered.
- This allows adding new functionalities into existing interfaces without breaking old implementations e.g. Collection, Comparator, ...

```
interface Emp {
    double getSal();
    default double calcIncentives() {
        return 0.0;
    }
}
class Manager implements Emp {
    // ...
    // calcIncentives() is overridden
    double calcIncentives() {
        return getSal() * 0.2;
    }
}
class Clerk implements Emp {
    // ...
    // calcIncentives() is not overridden -- so method of interface is
considered
}
```

```
new Manager().calcIncentives(); // return sal * 0.2
new Clerk().calcIncentives(); // return 0.0
```

- However default methods will lead to ambiguity errors as well, if same default method is available from multiple interfaces. Error: Duplicate method while declaring class.
- Superclass same method get higher priority. But super-interfaces same method will lead to error.
  - Super-class wins! Super-interfaces clash!!

```java
interface Displayable {
    default void show() {
        System.out.println("Displayable.show() called");
    }
}
interface Printable {
    default void show() {
        System.out.println("Printable.show() called");
    }
}
class FirstClass implements Displayable, Printable { // compiler error:
duplicate method
    // ...
}
class Main {
    public static void main(String[] args) {
        FirstClass obj = new FirstClass();
        obj.show();
    }
}
```

```java
interface Displayable {
    default void show() {
        System.out.println("Displayable.show() called");
    }
}
interface Printable {
    default void show() {
        System.out.println("Printable.show() called");
    }
}
class Superclass {
    public void show() {
        System.out.println("Superclass.show() called");
    }
}
class SecondClass extends Superclass implements Displayable, Printable {
    // ...
}
class Main {
    public static void main(String[] args) {
        SecondClass obj = new SecondClass();
        obj.show(); // Superclass.show() called
    }
}
```

- A class can invoke methods of super interfaces using InterfaceName.super.

```java
interface Displayable {
    default void show() {
        System.out.println("Displayable.show() called");
    }
}
interface Printable {
    default void show() {
        System.out.println("Printable.show() called");
    }
}
class FourthClass implements Displayable, Printable {
    @Override
    public  void show() {
        System.out.println("FourthClass.show() called");
        Displayable.super.show();
        Printable.super.show();
    }
}
class Main {
    public static void main(String[] args) {
        FourthClass obj = new FourthClass();
        obj.show(); // calls FourthClass method
    }
}
```

Static methods

- Before Java 8, interfaces allowed public static final fields.
- Java 8 also allows the static methods in interfaces.
- They act as helper methods and thus eliminates need of helper classes like Collections, ...

```java
interface Emp {
    double getSal();
    public static double calcTotalSalary(Emp[] a) {
        double total = 0.0;
        for(int i=0; i<a.length; i++)
            total += a[i].getSal();
        return total;
    }
}
```

Functional Interface

- If interface contains exactly one abstract method (SAM), it is said to be functional interface.
- It may contain additional default & static methods. E.g. Comparator, Runnable, ...
- @FunctionalInterface annotation does compile time check, whether interface contains single abstract method. If not, raise compile time error.

```java
@FunctionalInterface // okay
interface Foo {
    void foo(); // SAM
}
```

```java
@FunctionalInterface // okay
interface FooBar1 {
    void foo(); // SAM
    default void bar() {
        /*... */
    }
}
```

```java
@FunctionalInterface // NO -- error
interface FooBar2 {
    void foo(); // AM
    void bar(); // AM
}
```

```java
@FunctionalInterface // NO -- error
interface FooBar3 {
    default void foo() {
        /*... */
    }
    default void bar() {
        /*... */
    }
}
```

```java
@FunctionalInterface    // okay
interface FooBar4 {
    void foo(); // SAM
    public static void bar() {
        /*... */
    }
}
```

- Functional interfaces forms foundation for Java lambda expressions and method references.

Built-in functional interfaces

- New set of functional interfaces given in java.util.function package.

  - `Predicate<T>`: test: T -> boolean
  - `Function<T,R>`: apply: T -> R
  - `BiFunction<T,U,R>`: apply: (T,U) -> R
  - `UnaryOperator<T>`: apply: T -> T
  - `BinaryOperator<T>`: apply: (T,T) -> T
  - `Consumer<T>`: accept: T -> void
  - `Supplier<T>`: get: () -> T
- For efficiency primitive type functional interfaces are also supported e.g. IntPredicate, IntConsumer, IntSupplier, IntToDoubleFunction, ToIntFunction, ToIntBiFunction, IntUnaryOperator, IntBinaryOperator.

Lambda expressions

- Traditionally Java uses anonymous inner classes to compact the code. For each inner class separate .class file is created.
- However code is complex to read and un-efficient to execute.
- Lambda expression is short-hand way of implementing functional interface.
- Its argument types may or may not be given. The types will be inferred.
- Lambda expression can be single liner (expression not statement) or multi-liner block { ... }.

```
// Anonymous inner class
Arrays.sort(arr, new Comparator<Emp>() {
    public int compare(Emp e1, Emp e2) {
        int diff = e1.getEmpno() - e2.getEmpno();
        return diff;
    }
});
```

```
// Lambda expression -- multi-liner
Arrays.sort(arr, (Emp e1, Emp e2) -> {
    int diff = e1.getEmpno() - e2.getEmpno();
    return diff;
});
```

```
// Lambda expression -- multi-liner -- Argument types inferred
Arrays.sort(arr, (e1, e2) -> {
    int diff = e1.getEmpno() - e2.getEmpno();
    return diff;
});
```

```
// Lambda expression -- single-liner -- with block { ... }
Arrays.sort(arr, (e1, e2) -> {
    return e1.getEmpno() - e2.getEmpno();
});
```

```
// Lambda expression -- single-liner
Arrays.sort(arr, (e1,e2) -> e1.getEmpno() - e2.getEmpno());
```

- Practically lambda expressions are used to pass as argument to various functions.
- Lambda expression enable developers to write concise code (single liners recommended).

Non-capturing lambda expression

- If lambda expression result entirely depends on the arguments passed to it, then it is non-capturing (self-contained).

```
BinaryOperator<Integer> op1 = (a,b) -> a + b;
testMethod(op);
```

```
static void testMethod(BinaryOperator<Integer> op) {
    int x=12, y=5, res;
    res = op.apply(x, y); // res = x + y;
    System.out.println("Result: " + res)
}
```

- In functional programming, such functions/lambdas are referred as pure functions.

Capturing lambda expression

- If lambda expression result also depends on additional variables in the context of the lambda expression passed to it, then it is capturing.

```
int c = 2; // must be effectively final
BinaryOperator<Integer> op = (a,b) -> a + b + c;
testMethod(op);
```

```
static void testMethod(BinaryOperator<Integer> op) {
    int x=12, y=5, res;
    res = op.apply(x, y); // res = x + y + c;
    System.out.println("Result: " + res);
}
```

- Here variable c is bound (captured) into lambda expression. So it can be accessed even out of scope (effectively). Internally it is associated with the method/expression.
- In some functional languages, this is known as Closures.

Java 8 Streams

- Java 8 Stream is NOT IO streams.
- java.util.stream package.
- Streams follow functional programming model in Java 8.
- The functional programming is based on functional interface (SAM).
- Number of predefined functional interfaces added in Java 8. e.g. Consumer, Supplier, Function, Predicate, ...
- Lambda expression is short-hand way of implementing SAM -- arg types & return type are inferred.
- Java streams represents pipeline of operations through which data is processed.
- Stream operations are of two types
    - Intermediate operations: Yields another stream.
        - filter()
        - map(), flatMap()
        - limit(), skip()
        - sorted(), distinct()
    - Terminal operations: Yields some result.
        - reduce()
        - forEach()
        - collect(), toArray()
        - count(), max(), min()
    - Stream operations are higher order functions (take functional interfaces as arg).

Java stream characteristics

- No storage: Stream is an abstraction. Stream doesn't store the data elements. They are stored in source collection or produced at runtime.
- Immutable: Any operation doesn't change the stream itself. The operations produce new stream of results.
- Lazy evaluation: Stream is evaluated only if they have terminal operation. If terminal operation is not given, stream is not processed.
- Not reusable: Streams processed once (terminal operation) cannot be processed again.

Stream creation

- Collection interface: stream() or parallelStream()
- Arrays class: Arrays.stream()
- Stream interface: static of() method
- Stream interface: static generate() method
- Stream interface: static iterate() method
- Stream interface: static empty() method
- nio Files class: `static Stream<String> lines(filePath)` method

Stream creation

- Collection interface: stream() or parallelStream()

```
List<String> list = new ArrayList<>();
// ...
Stream<String> strm = list.stream();
```

- Arrays class: Arrays.stream()
- Stream interface: static of() method

```
Stream<Integer> strm = Stream.of(arr);
```

- Stream interface: static generate() method
  - generate() internally calls given Supplier in an infinite loop to produce infinite stream of elements.

```
Stream<Double> strm = Stream.generate(() -> Math.random()).limit(25);
```

```
Random r = new Random();
Stream<Integer> strm = Stream.generate(() -> r.nextInt(1000)).limit(10);
```

- Stream interface: static iterate() method
  - iterate() start the stream from given (arg1) "seed" and calls the given UnaryOperator in infinite loop to produce infinite stream of elements.

```
Stream<Integer> strm = Stream.iterate(1, i -> i + 1).limit(10);
```

- Stream interface: static empty() method
- nio Files class: static Stream lines(filePath) method

Stream operations

- Source of elements

```
String[] names = {"Smita", "Rahul", "Rachana", "Amit", "Shraddha", "Nilesh",
"Rohan", "Pradnya", "Rohan", "Pooja", "Lalita"};
```

- Create Stream and display all names

```
Stream.of(names)
    .forEach(s -> System.out.println(s));
```

- filter() -- Get all names ending with "a"
  - `Predicate<T>`: (T) -> boolean

```
Stream.of(names)
    .filter(s -> s.endsWith("a"))
    .forEach(s -> System.out.println(s));
```

- map() -- Convert all names into upper case
  - `Function<T,R>`: (T) -> R

```
Stream.of(names)
    .map(s -> s.toUpperCase())
    .forEach(s -> System.out.println(s));
```

- sorted() -- sort all names in ascending order
  - String class natural ordering is ascending order.
  - sorted() is a stateful operation (i.e. needs all element to sort).

```
Stream.of(names)
    .sorted()
    .forEach(s -> System.out.println(s));
```

- sorted() -- sort all names in descending order
  - `Comparator<T>`: (T,T) -> int

```
Stream.of(names)
    .sorted((x,y) -> y.compareTo(x))
    .forEach(s -> System.out.println(s));
```

- skip() & limit() -- leave first 2 names and print next 4 names

```
Stream.of(names)
    .skip(2)
    .limit(4)
    .forEach(s -> System.out.println(s));
```

- distinct() -- remove duplicate names
  - duplicates are removed according to equals().

```
Stream.of(names)
    .distinct()
    .forEach(s -> System.out.println(s));
```

- count() -- count number of names
  - terminal operation: returns long.

```
long cnt = Stream.of(names)
    .count();
System.out.println(cnt);
```

- collect() -- collects all stream elements into an collection (list, set, or map)

```
List<String> list = Stream.of(names)
        .collect(Collectors.toList());
// Collectors.toList() returns a Collector that can collect all stream
elements into a list
```

```
Set<String> set = Stream.of(names)
    .collect(Collectors.toSet());
// Collectors.toSet() returns a Collector that can collect all stream
elements into a set
```

- reduce() -- addition of 1 to 5 numbers

```
int result = Stream
    .iterate(1, i -> i+1)
    .limit(5)
    .reduce(0, (x,y) -> x + y);
```

- max() -- find the max string
  - terminal operation
  - See examples.

Default methods

day15 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CI-H-02 main]
  - JRE System Library [JavaSE-1.8]
  - src
    - com.sunbeam
      - Circle.java
      - Program01.java
      - Rectangle.java
      - Shape.java
      - Square.java
- demo02 [CI-H-02 main]
  - JRE System Library [JavaSE-1.8]
  - src
    - com.sunbeam
      - Program02.java

Program02.java

```java
package com.sunbeam;

interface Printable {
    void show();
}
interface Displayable {
    void show();
}
class MyClass implements Printable, Displayable {
    @Override
    public void show() {
        System.out.println("MyClass.show() called.");
    }
}

public class Program02 {
    public static void main(String[] args) {
        MyClass obj1 = new MyClass();
        obj1.show();
        Printable obj2 = new MyClass();
        obj2.show();
        Displayable obj3 = new MyClass();
        obj3.show();
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELE

```
MyClass.show() called.
MyClass.show() called.
MyClass.show() called.
```

Search 10:18 AM

day15 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
interface Printable {
    default void show() {
        System.out.println("Printable.show() called.");
    }
}
interface Displayable {
    default void show() {
        System.out.println("Displayable.show() called.");
    }
}
// if two interfaces have default method with same signature and a class is inher
//  then it will lead to ambiguity.
// this problem can be resolved by overriding method in sub-class.
class SecondClass implements Printable, Displayable {
    public void show() {
        System.out.println("SecondClass.show() called.");
    }
}
public class Program02 {
    public static void main(String[] args) {
        SecondClass obj1 = new SecondClass();
        obj1.show();
        Printable obj2 = new SecondClass();
        obj2.show();
        Displayable obj3 = new SecondClass();
        obj3.show();
```

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELE

```
SecondClass.show() called.
SecondClass.show() called.
SecondClass.show() called.
```

Search 10:25 AM

day15 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
class Printable {
    public void show() {
        System.out.println("Printable.show() called.");
    }
}
interface Displayable {
    default void show() {
        System.out.println("Displayable.show() called.");
    }
}
// when same signature method is inherited from a super-class
// and a super-interface, the super-class method gets precedance.
// no compiler error for ambiguity
class ThirdClass extends Printable implements Displayable {
}

public class Program02 {
    public static void main(String[] args) {
        ThirdClass obj1 = new ThirdClass();
        obj1.show();
        Printable obj2 = new ThirdClass();
        obj2.show();
        Displayable obj3 = new ThirdClass();
        obj3.show();
    }
}
```

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELE

```
Printable.show() called.
Printable.show() called.
Printable.show() called.
```

super-class wins! super-interfaces clash!!

Search 10:28 AM

day15 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
114
115 class Printable {
116     public void show() {
117         System.out.println("Printable.show() called.");
118     }
119 }
120 interface Displayable {
121     default void show() {
122         System.out.println("Displayable.show() called.");
123     }
124 }
125 // method overrriding -- method is called depending on type of object.
126 class FourthClass extends Printable implements Displayable {
127     public void show() {
128         System.out.println("FourthClass.show() called.");
129     }
130 }
131 public class Program02 {
132     public static void main(String[] args) {
133         FourthClass obj1 = new FourthClass();
134         obj1.show();
135         Printable obj2 = new FourthClass();
136         obj2.show();
137         Displayable obj3 = new FourthClass();
138         obj3.show();
139     }
```

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELE

```
FourthClass.show() called.
FourthClass.show() called.
FourthClass.show() called.
```

Search 10:31 AM

day15 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program02.java

```java
interface Printable {
    default void show() {
        System.out.println("Printable.show() called.");
    }
}
interface Displayable {
    default void show() {
        System.out.println("Displayable.show() called.");
    }
}
// method overrriding -- method is called depending on type of object.
class FourthClass implements Printable, Displayable {
    public void show() {
        System.out.println("FourthClass.show() called.");
        // default methods in super interface can be called from sub-class methods
        //super.show(); // error: show() is not found in super-class i.e. Object
        Printable.super.show();
        Displayable.super.show();
    }
}
public class Program02 {
    public static void main(String[] args) {
        FourthClass obj1 = new FourthClass();
        obj1.show();
    }
}
```

Problems Javadoc Declaration Console

<terminated> Program02 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELE

```
FourthClass.show() called.
Printable.show() called.
Displayable.show() called.
```

Search 10:38 AM

File Edit Selection View Go Run ··· private

day15.md × day14.md classwork.md U

day15.md > # Core Java > ## Java 8 Interfaces > #### Functional Interface

Functional Interface

* If interface contains exactly one abstract method (SAM), it is said to be functional interface.
* It may contain additional default & static methods. E.g. Comparator, Runnable, …
* @FunctionalInterface annotation does compile time check, whether interface contains single abstract method. If not, raise compile time error.

```Java
@FunctionalInterface // okay
interface Foo {
    void foo(); // SAM
}
```

```java
@FunctionalInterface // okay
interface FooBar1 {
    void foo(); // SAM
    default void bar() {
        /*... */
    }
```

pre-defined functional interface:

1. Comparable
2. Comparator
3. Runnable
4. Closeable/AutoCloseable

java.util.function package -- since Java 8:

- Predicate, Function, BinaryOperator, Supplier, Consumer, ...

main* 0 0 0 Ln 218, Col 36 Spaces: 4 UTF-8 CRLF Markdown

Search 10:48 AM

day15 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java × Employee.java

```java
public class Program03 {
    public static void main(String[] args) {
        Employee[] arr = new Employee[] {
            new Employee(4, "B", "Clerk", "Sales", 723.44),
            new Employee(8, "X", "Manager", "Accounts", 823.23),
            new Employee(2, "P", "Clerk", "Research", 234.23),
            new Employee(9, "N", "Manger", "Sales", 252.53),
            new Employee(5, "D", "Clerk", "Accounts", 923.23),
            new Employee(1, "Q", "Analyst", "Research", 826.23),
            new Employee(7, "H", "Clerk", "Research", 845.24),
            new Employee(6, "A", "Analyst", "Research", 832.23),
            new Employee(3, "G", "Analyst", "Sales", 952.44)
        };

        System.out.println("Emps sorted by id -- using EmpIdComparator -- local class");
        class EmpIdComparator implements Comparator<Employee> {
            @Override
            public int compare(Employee x, Employee y) {
                int diff = x.getId() - y.getId(); //Integer.compare(x.getId(), y.getId());
                return diff;
            }
        }
        Arrays.sort(arr, new EmpIdComparator());
        for (Employee e : arr)
            System.out.println(e);
    }
}
```

Problems Javadoc Declaration Console ×

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELE

```
Emps sorted by id -- using EmpIdCompar
Employee [id=1, name=Q, job=Analyst, d
Employee [id=2, name=P, job=Clerk, dep
Employee [id=3, name=G, job=Analyst, d
Employee [id=4, name=B, job=Clerk, dep
Employee [id=5, name=D, job=Clerk, dep
Employee [id=6, name=A, job=Analyst, d
Employee [id=7, name=H, job=Clerk, dep
Employee [id=8, name=X, job=Manager, d
Employee [id=9, name=N, job=Manger, de
```

Search 11:04 AM

day15 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java | Employee.java

```
        System.out.println("Emps sorted by id -- using EmpIdComparator -- local class");
        class EmpIdComparator implements Comparator<Employee> {
            @Override
            public int compare(Employee x, Employee y) {
                int diff = x.getId() - y.getId(); //Integer.compare(x.getId(), y.getId());
                return diff;
            }
        }
        Arrays.sort(arr, new EmpIdComparator());
        for (Employee e : arr)
            System.out.println(e);

        System.out.println("\nEmps sorted by name -- using Anonymous Inner class");
        Comparator<Employee> empNameComparator = new Comparator<Employee>() {
            @Override
            public int compare(Employee x, Employee y) {
                int diff = x.getName().compareTo(y.getName());
                return diff;
            }
        };
        Arrays.sort(arr, empNameComparator);
        for (Employee e : arr)
            System.out.println(e);
    }
}
```

impl of $1 (anon cls)

new ClassName() {
    // ...
};

Comparator<Emp>

$1

$1

Writable | Smart Insert | 32 : 31 : 1264

Search | 11:32 AM

day15 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java | Employee.java

```
31
32          System.out.println("\nEmps sorted by name -- using Anonymou
33          Comparator<Employee> empNameComparator = new Comparator<Em
34              @Override
35              public int compare(Employee x, Employee y) {
36                  int diff = x.getName().compareTo(y.getName());
37                  return diff;
38              }
39          };
40          Arrays.sort(arr, empNameComparator);
41          for (Employee e : arr)
42              System.out.println(e);
43
44          System.out.println("\nEmps sorted by job -- using Anonymous Inner class Anonymous object");
45          Arrays.sort(arr, new Comparator<Employee>() {
46              @Override
47              public int compare(Employee x, Employee y) {
48                  int diff = x.getJob().compareTo(y.getJob());
49                  return diff;
50              }
51          });
52          for (Employee e : arr)
53              System.out.println(e);
54
55
56      }
```

Problems | Javadoc | Declaration | Console

<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.just

```
Emps sorted by job -- using Anonymous Inner class Anc
Employee [id=6, name=A, job=Analyst, dept=Research, s
Employee [id=3, name=G, job=Analyst, dept=Sales, sal=
Employee [id=1, name=Q, job=Analyst, dept=Research, s
Employee [id=4, name=B, job=Clerk, dept=Sales, sal=72
Employee [id=5, name=D, job=Clerk, dept=Accounts, sal
Employee [id=7, name=H, job=Clerk, dept=Research, sal
Employee [id=2, name=P, job=Clerk, dept=Research, sal
Employee [id=8, name=X, job=Manager, dept=Accounts, s
Employee [id=9, name=N, job=Manger, dept=Sales, sal=2
```

Anonymous object of Anoymous class passed as 2nd arg in sort() method.

Comparator<Employee>

$2

Search 11:40 AM

day15 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java | Employee.java

```java
        System.out.println("\nEmps sorted by job -- using Anonymous
        Arrays.sort(arr, new Comparator<Employee>() {
            @Override
            public int compare(Employee x, Employee y) {
                int diff = x.getJob().compareTo(y.getJob());
                return diff;
            }
        });
        for (Employee e : arr)
            System.out.println(e);

        System.out.println("\nEmps sorted by job in desc order: ");
        Arrays.sort(arr, (Employee x, Employee y) -> {
            int diff = -x.getJob().compareTo(y.getJob());
            return diff;
        });
        for (Employee e : arr)
            System.out.println(e);
    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.just
Emps sorted by job in desc order:
Employee [id=9, name=N, job=Manger, dept=Sales, sal=2
Employee [id=8, name=X, job=Manager, dept=Accounts, s
Employee [id=4, name=B, job=Clerk, dept=Sales, sal=72
Employee [id=5, name=D, job=Clerk, dept=Accounts, sal
Employee [id=7, name=H, job=Clerk, dept=Research, sal
Employee [id=2, name=P, job=Clerk, dept=Research, sal
Employee [id=6, name=A, job=Analyst, dept=Research, s
Employee [id=3, name=G, job=Analyst, dept=Sales, sal=
Employee [id=1, name=Q, job=Analyst, dept=Research, s
```

Lambda expression is short-hand implementation of the abstract method in the functional interface.

Search 11:51 AM

day15 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java | Employee.java

```
        });
        for (Employee e : arr)
            System.out.println(e);

        System.out.println("\nEmps sorted by sal in asc order: ");
        Arrays.sort(arr, (x, y) -> {
            return Double.compare(x.getSal(), y.getSal());
        });
        for (Employee e : arr)
            System.out.println(e);

        System.out.println("\nEmps sorted by sal in desc order: ");
        // single liner lambda expression doesn't need curly braces
        // and the result of expression is considered to be returned.
        Arrays.sort(arr, (x, y) -> - Double.compare(x.getSal(), y.getSal()) );
        for (Employee e : arr)
            System.out.println(e);
    }
}
```

single abstract method → of the functional interafce

arguments

one-liner implementatin

Problems | Javadoc | Declaration | Console

```
<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.just
id=3, name=G, job=Analyst, dept=Sales, sal=952.44]
id=5, name=D, job=Clerk, dept=Accounts, sal=923.23]
id=7, name=H, job=Clerk, dept=Research, sal=845.24]
id=6, name=A, job=Analyst, dept=Research, sal=832.23]
id=1, name=Q, job=Analyst, dept=Research, sal=826.23]
id=8, name=X, job=Manager, dept=Accounts, sal=823.23]
id=4, name=B, job=Clerk, dept=Sales, sal=723.44]
id=9, name=N, job=Manger, dept=Sales, sal=252.53]
id=2, name=P, job=Clerk, dept=Research, sal=234.23]
```

Search 12:00 PM

day15 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java | Employee.java

```
79              System.out.println(e);
80
81          System.out.println("\nEmps sorted by sal in desc order: "
82          // single liner lambda expression doesn't need curly brac
83          // and the result of expression is considered to be retur
84          Arrays.sort(arr, (x, y) -> - Double.compare(x.getSal(), y
85          for (Employee e : arr)
86              System.out.println(e);
87
88          System.out.println("\nEmps list sorted by id in desc orde
89          List<Employee> list = Arrays.asList(arr);
90          list.sort((x,y) -> -Integer.compare(x.getId(), y.getId()),
91          list.forEach(e -> System.out.println(e.toString()));
92      }
93  }
94
95
96
97
98
99
100
101
102
103
104
```

Problems | Javadoc | Declaration | Console

```
<terminated> Program03 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\plugins\org.eclipse.just
Emps list sorted by id in desc order:
Employee [id=9, name=N, job=Manger, dept=Sales, sal=2
Employee [id=8, name=X, job=Manager, dept=Accounts, s
Employee [id=7, name=H, job=Clerk, dept=Research, sal
Employee [id=6, name=A, job=Analyst, dept=Research, s
Employee [id=5, name=D, job=Clerk, dept=Accounts, sal
Employee [id=4, name=B, job=Clerk, dept=Sales, sal=72
Employee [id=3, name=G, job=Analyst, dept=Sales, sal=
Employee [id=2, name=P, job=Clerk, dept=Research, sal
Employee [id=1, name=Q, job=Analyst, dept=Research, s
```

Search 12:10 PM

day15 - demo03/src/com/sunbeam/Program03.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program03.java | Employee.java

```java
        });
        for (Employee e : arr)
            System.out.println(e);

        System.out.println("\nEmps sorted by sal in asc order: ");
        Arrays.sort(arr, (x, y) -> {
            return Double.compare(x.getSal(), y.getSal());
        });
        for (Employee e : arr)
            System.out.println(e);

        System.out.println("\nEmps sorted by sal in desc order: ");
        // single liner lambda expression doesn't need curly braces
        // and the result of expression is considered to be returned.
        Arrays.sort(arr, (x, y) -> - Double.compare(x.getSal(), y.getSal()) );
        for (Employee e : arr)
            System.out.println(e);

        System.out.println("\nEmps list sorted by id in desc order: ");
        List<Employee> list = Arrays.asList(arr);
        list.sort((x,y) -> -Integer.compare(x.getId(), y.getId()));
        list.forEach(e -> System.out.println(e.toString()));
    }
}
```

Lambda expressions are executed with a special byte-code instruction i.e. "invokedynamic"

Writable | Smart Insert | 93 : 2 : 3538

Search | 12:17 PM

day15 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
package com.sunbeam;

import java.util.function.BinaryOperator;

public class Program04 {
    // lambda expressions are referenced by functional interface reference.
    // lambda arg scope is limited to lambda expression body/implementation.
    //
    public static void main(String[] args) {
        // non-capturing lambda
        BinaryOperator<Integer> op1 = (x,y) -> x + y;

        // capturing lambda - captures (attach) a variable out-side the lambda implementation.
        // can capture variables that are final or effectively final.
        int z = 10; // "z" is captured in lambda "op2"
        BinaryOperator<Integer> op2 = (x,y) -> x + y + z;
        //z++; // if z is modified, it cannot be captured into the lambda expression.

        int a = 22, b = 7;
        int r = op1.apply(a, b);
        System.out.println("op1 result: " + r); // 29

        r = op2.apply(a, b);
        System.out.println("op2 result: " + r); // 39
    }
}
```

"invokedynamic" → a + b

29 = 22 + 7

"invokedynamic" → a + b + z

39 = 22 + 7 + 10

Problems | Javadoc | Declaration | Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\pl

```
op1 result: 29
op2 result: 39
```

Search 12:32 PM

day15 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
        //z++; // if z is modified, it cannot be captured into the lambda expr

        int a = 22, b = 7;
        int r = op1.apply(a, b);
        System.out.println("op1 result: " + r); // 29

        r = op2.apply(a, b);
        System.out.println("op2 result: " + r); // 39

        calc(20, 10, (x,y) -> x * y);
    }

    public static void calc(int n1, int n2, BinaryOperator<Integer> op) {
        int res = op.apply(n1, n2);-- invokedynamic -- n1 * n2
        System.out.println("Result: " + res);
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\pl

```
op2 result: 39
Result: 200
```

Search 12:41 PM

day15 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
        // can capture variables that are final or effectively final.
        int z = 10; // "z" is captured in lambda "op2"
        BinaryOperator<Integer> op2 = (x,y) -> x + y + z;
        //z++; // if z is modified, it cannot be captured into the lambda expr

        int a = 22, b = 7;
        int r = op1.apply(a, b);
        System.out.println("op1 result: " + r); // 29

        r = op2.apply(a, b);
        System.out.println("op2 result: " + r); // 39

        calc(20, 10, (x,y) -> x * y);

        calc(20, 10, (x,y) -> x * y * z);
    }

    public static void calc(int n1, int n2, BinaryOperator<Integer> op) {
        int res = op.apply(n1, n2);
        System.out.println("Result: " + res);
    }
}
```

invokedynamic --> n1 * n2 * z;

Problems | Javadoc | Declaration | Console

<terminated> Program04 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\pl

```
Result: 200
Result: 2000
```

Capturing Lambdas are also called as "Closure" in few programming languages.

Search 12:43 PM

day15 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java

```java
    public static void main(String[] args) {
        // Input: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
        // Step1: square each number : 1, 4, 9, 16, 25, 36, 49, 64, 81, 100
        // Step2: get all odd numbers: 1, 9, 25, 49, 81
        // Step3: prefix with "Java" : "Java1", "Java9", "Java25", "Java49",
        // Output: print each element

        Stream<Integer> strm1 = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
            // 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
        Stream<Integer> strm2 = strm1.map(n -> n * n);
            // 1, 4, 9, 16, 25, 36, 49, 64, 81, 100
        Stream<Integer> strm3 = strm2.filter(n -> n % 2 != 0);
            // 1, 9, 25, 49, 81
        Stream<String> strm4 = strm3.map(n -> "Java"+n);
            // "Java1", "Java9", "Java25", "Java49", "Java81"
        strm4.forEach(s -> System.out.println(s));
    }   strm4.collect(...); <-- IllegalStateException
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> Program05 [Java Application] C:\Nilesh\setup\sts-4.15.1.RELEASE\pl
Java1
Java9
Java25
Java49
Java81
```

Stream Operations

Intermediate Operations

- returns a new Stream

1. map()
2. filter()
3. limit()
4. skip()
5. flatMap()
6. sorted()
7. ...

Terminal Operations

- returns non-Stream

1. forEach()
2. reduce()
3. collect()
4. ...

Stream characteristics:

1. Immutable
2. Not reusable -- only one terminal operation.
3. No storage -- is not a collection (have temp memory)
4. Lazily evaluated -- all ops work only if terminal operation is given.

Search 1:09 PM

day15 - demo05/src/com/sunbeam/Program05.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program05.java

```java
    }
    */

    public static void main(String[] args) {
        Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
            .map(n -> {
                System.out.println("map() -- square -- " + n);
                return n * n;
            })
            .filter(n -> {
                System.out.println("filter() -- odd nums -- " + n);
                return n % 2 != 0;
            })
            .sorted((x,y) -> {
                System.out.println("sorted() -- " + x + " <> " + y);
                return y - x; // desc sort
            })
            .map(n -> {
                System.out.println("map() -- prefix Java -- " + n);;
                return "Java"+n;
            })
            .forEach(s -> System.out.println("forEach() -- " + s));
        System.out.println("Bye!");
    }
}
```

Writable | Smart Insert | 84 : 1 : 2472

Search | 1:32 PM

Core Java

Collect Stream result

- Collecting stream result is terminal operation.
- Object[] toArrray()
- R collect(Collector)
    - Collectors.toList(), Collectors.toSet(), Collectors.toCollection(), Collectors.joining()
    - Collectors.toMap(key, value)

Stream of primitive types

- Efficient in terms of storage and processing. No auto-boxing and unboxing is done.
- IntStream class
    - IntStream.of() or IntStream.range() or IntStream.rangeClosed() or Random.ints()
    - sum(), min(), max(), average(), summaryStatistics(),

Method references

- If lambda expression involves single method call, it can be shortened by using method reference.
- Method references are converted into instances of functional interfaces.
- Method reference can be used for class static method, class non-static method, object non-static method or constructor.

Examples

- Class static method: Integer::sum [(a,b) -> Integer.sum(a,b)]
    - Both lambda param passed to static function explicitly
- Class non-static method: String::compareTo [(a,b) -> a.compareTo(b)]
    - First lambda param become implicit param (this) of the function and second is passed explicitly (as arguments).
- Object non-static method: System.out::println [x -> System.out.println(x)]
    - Lambda param is passed to function explicitly.
- Constructor: Date::new [() -> new Date()]
    - Lambda param is passed to constructor explicitly.

enum

- "enum" keyword is added in Java 5.0.
- Used to make constants to make code more readable.
- Typical switch case

```
int choice;
// ...
switch(choice) {
    case 1: // addition
        c = a + b;
```

```
        break;
    case 2: // subtraction
        c = a - b;
        break;
    // ...
}
```

- The switch constants can be made more readable using Java enums.

```
enum ArithmeticOperations {
    ADDITION, SUBTRACTION, MULIPLICATION, DIVISION;
}

ArithmeticOperations choice = ArithmeticOperations.ADDITION;
// ...
switch(choice) {
    case ADDITION:
        c = a + b;
        break;
    case SUBTRACTION:
        c = a - b;
        break;
    // ...
}
```

- In java, enums cannot be declared locally (within a method).
- The declared enum is converted into enum class.

```
// user-defined enum
enum ArithmeticOperations {
    ADDITION, SUBTRACTION, MULIPLICATION, DIVISION;
}
```

```
// generated enum code
final class ArithmeticOperations extends Enum {
    public static ArithmeticOperations[] values() {
        return (ArithmeticOperations[])$VALUES.clone();
    }
    public static ArithmeticOperations valueOf(String s) {
        return (ArithmeticOperations)Enum.valueOf(ArithmeticOperations, s);
    }
    private ArithmeticOperations(String name, int ordinal) {
        super(name, ordinal); // invoke sole constructor Enum(String,int);
    }
    public static final ArithmeticOperations ADDITION;
    public static final ArithmeticOperations SUBTRACTION;
    public static final ArithmeticOperations MULIPLICATION;
```

```
    public static final ArithmeticOperations DIVISION;
    private static final ArithmeticOperations $VALUES[];
    static {
        ADDITION = new ArithmeticOperations("ADDITION", 0);
        SUBTRACTION = new ArithmeticOperations("SUBTRACTION", 1);
        MULIPLICATION = new ArithmeticOperations("MULIPLICATION", 2);
        DIVISION = new ArithmeticOperations("DIVISION", 3);
        $VALUES = (new ArithmeticOperations[] {
            ADDITION, SUBTRACTION, MULIPLICATION, DIVISION
        });
    }
}
```

- The enum type declared is implicitly inherited from java.lang.Enum class. So it cannot be extended from another class, but enum may implement interfaces.
- The enum constants declared in enum are public static final fields of generated class. Enum objects cannot be created explicitly (as generated constructor is private).
- The generated class will have a values() method that returns array of all constants and valueOf() method to convert String to enum constant.
- The enums constants can be used in switch-case and can also be compared using == operator.
- The java.lang.Enum class has following members:

```
public abstract class Enum<E> implements java.lang.Comparable<E>,
java.io.Serializable {
    private final String name;
    private final int ordinal;

    protected Enum(String,int); // sole constructor - can be called from
user-defined enum class only
    public final String name(); // name of enum const
    public final int ordinal(); // position of enum const (0-based)

    public String toString(); // returns name of const
    public final int compareTo(E);// compares with another enum of same type
on basis of ordinal number
    public static <T> T valueOf(Class<T>, String);
    // ...
}
```

- The enum may have fields and methods.

```
enum Element {
    H(1, "Hydrogen"),
    HE(2, "Helium"),
    LI(3, "Lithium");

    public final int num;
    public final String label;
```

```
    private Element(int num, String label) {
        this.num = num;
        this.label = label;
    }
}
```

Reflection

- .class = Byte-code + Meta-data + Constant pool + ...
- When class is loaded into JVM all the metadata is stored in the object of java.lang.Class (heap area).
- This metadata includes class name, super class, super interfaces, fields (field name, field type, access modifier, flags), methods (method name, method return type, access modifier, flags, method arguments, ...), constructors (access modifier, flags, ctor arguments, ...), annotations (on class, fields, methods, ...).

Reflection applications

- Inspect the metadata (like javap)
- Build IDE/tools (Intellisense)
- Dynamically creating objects and invoking methods
- Access the private members of the class

Get the java.lang.Class object

- way 1: When you have class-name as a String (taken from user or in properties file)

  ```
  Class<?> c = Class.forName(className);
  ```

- way 2: When the class is in project/classpath.

  ```
  Class<?> c = ClassName.class;
  ```

- way 3: When you have object of the class.

  ```
  Class<?> c = obj.getClass();
  ```

Access metadata in java.lang.Class

- Name of the class

  ```
  String name = c.getName();
  ```

- Super class of the class

```
Class<?> supcls = c.getSuperclass();
```

- Super interfaces of the class

```
Class<?> supintf[] = c.getInterfaces();
```

- Fields of the class

```
Field[] fields = c.getFields(); // all fields accessible (of class & its
super class)
```

```
Field[] fields = c.getDeclaredFields(); // all fields in the class
```

- Methods of the class

```
Method[] methods = c.getMethods(); // all methods accessible (of class & its
super class)
```

```
Method[] methods = c.getDeclaredMethods(); // all methods in the class
```

- Constructors of the class

```
Constructor[] ctors = c.getConstructors(); // all ctors accessible (of class
& its super class)
```

```
Constructor[] ctors = c.getDeclaredConstructor(); // all ctors in the class
```

Reflection Tutorial

- https://youtu.be/lAoNJ_7LD44
- https://youtu.be/UVWdtk5ibK8

Annotations

- Added in Java 5.0.
- Annotation is a way to associate metadata with the class and/or its members.
- Annotation applications
    - Information to the compiler
    - Compile-time/Deploy-time processing
    - Runtime processing
- Annotation Types
    - Marker Annotation: Annotation is not having any attributes.
        - @Override, @Deprecated, @FunctionalInterface ...
    - Single value Annotation: Annotation is having single attribute -- usually it is "value".
        - @SuppressWarnings("deprecation"), ...
    - Multi value Annotation: Annotation is having multiple attribute
        - @RequestMapping(method = "GET", value = "/books"), ...

Pre-defined Annotations

- @Override
    - Ask compiler to check if corresponding method (with same signature) is present in super class.
    - If not present, raise compiler error.
- @FunctionalInterface
    - Ask compiler to check if interface contains single abstract method.
    - If zero or multiple abstract methods, raise compiler error.
- @Deprecated
    - Inform compiler to give a warning when the deprecated type/member is used.
- @SuppressWarnings
    - Inform compiler not to give certain warnings: e.g. deprecation, rawtypes, unchecked, serial, unused
    - @SuppressWarnings("deprecation")
    - @SuppressWarnings({"rawtypes", "unchecked"})
    - @SuppressWarnings("serial")
    - @SuppressWarnings("unused")

Meta-Annotations

- Annotations that apply to other annotations are called meta-annotations.
- Meta-annotation types defined in java.lang.annotation package.

@Retention

- RetentionPolicy.SOURCE
    - Annotation is available only in source code and discarded by the compiler (like comments).
    - Not added into .class file.
    - Used to give information to the compiler.
    - e.g. @Override, ...
- RetentionPolicy.CLASS
    - Annotation is compiled and added into .class file.
    - Discared while class loading and not loaded into JVM memory.
    - Used for utilities that process .class files.

    - e.g. Obfuscation utilities can be informed not to change the name of certain class/member using @SerializedName, ...
- RetentionPolicy.RUNTIME
    - Annotation is compiled and added into .class file. Also loaded into JVM at runtime and available for reflective access.
    - Used by many Java frameworks.
    - e.g. @RequestMapping, @Id, @Table, @Controller, ...

@Target

- Where this annotation can be used.
- ANNOTATION_TYPE, CONSTRUCTOR, FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE, TYPE_PARAMETER, TYPE_USE
- If annotation is used on the other places than mentioned in @Target, then compiler raise error.

@Documented

- This annotation should be documented by javadoc or similar utilities.

@Repeatable

- The annotation can be repeated multiple times on the same class/target.

@Inherited

- The annotation gets inherited to the sub-class and accessible using c.getAnnotation() method.

Custom Annotation

- Annotation to associate developer information with the class and its members.

```
@Inherited
@Retention(RetentionPolicy.RUNTIME) // the def attribute is considered as
"value" = @Retention(value = RetentionPolicy.RUNTIME )
@Target({TYPE, CONSTRUCTOR, FIELD, METHOD}) // { } represents array
@interface Developer {
    String firstName();
    String lastName();
    String company() default "Sunbeam";
    String value() default "Software Engg";
}

@Repeatable
@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE})
@interface CodeType {
    String[] value();
}
```

```
//@Developer(firstName="Nilesh", lastName="Ghule", value="Technical
Director") // compiler error -- @Developer is not @Repeatable
@CodeType({"businessLogic", "algorithm"})
@Developer(firstName="Nilesh", lastName="Ghule", value="Technical Director")
class MyClass {
    // ...
    @Developer(firstName="Shubham", lastName="Patil", company="Sunbeam Karad
")
    private int myField;
    @Developer(firstName="Rahul", lastName="Sansuddi")
    public MyClass() {

    }
    @Developer(firstName="Shubham", lastName="Borle", company="Sunbeam Karad
")
    public void myMethod() {
        @Developer(firstName="James", lastName="Bond") // compiler error
        int localVar = 1;
    }
}
```

```
// @Developer is inherited
@CodeType("frontEnd")
@CodeType("businessLogic") // allowed because @CodeType is @Repeatable
class YourClass extends MyClass {
    // ...
}
```

Annotation tutorials

- Part 1: https://youtu.be/7zjWPJqlPRY
- Part 2: https://youtu.be/CafN2ABJQcg

Java IO framework

- Input/Output functionality in Java is provided under package java.io and java.nio package.
- IO framework is used for File IO, Network IO, Memory IO, and more.
- File is a collection of data and information on a storage device.
- File = Data + Metadata
- Two types of APIs are available file handling
    - FileSystem API -- Accessing/Manipulating Metadata
    - File IO API -- Accessing/Manipulating Contents/Data

java.io.File class

- A path (of file or directory) in file system is represented by "File" object.
- Used to access/manipulate metadata of the file/directory.
- Provides FileSystem APIs

    - String[] list() -- return contents of the directory
    - File[] listFiles() -- return contents of the directory
    - boolean exists() -- check if given path exists
    - boolean mkdir() -- create directory
    - boolean mkdirs() -- create directories (child + parents)
    - boolean createNewFile() -- create empty file
    - boolean delete() -- delete file/directory
    - boolean renameTo(File dest) -- rename file/directory
    - String getAbsolutePath() -- returns full path (drive:/folder/folder/...)
    - String getPath() -- return path
    - File getParentFile() -- returns parent directory of the file
    - String getParent() -- returns parent directory path of the file
    - String getName() -- return name of the file/directory
    - static File[] listRoots() -- returns all drives in the systems.
    - long getTotalSpace() -- returns total space of current drive
    - long getFreeSpace() -- returns free space of current drive
    - long getUsableSpace() -- returns usable space of current drive
    - boolean isDirectory() -- return true if it is a directory
    - boolean isFile() -- return true if it is a file
    - boolean isHidden() -- return true if the file is hidden
    - boolean canExecute()
    - boolean canRead()
    - boolean canWrite()
    - boolean setExecutable(boolean executable) -- make the file executable
    - boolean setReadable(boolean readable) -- make the file readable
    - boolean setWritable(boolean writable) -- make the file writable
    - long length() -- return size of the file in bytes
    - long lastModified() -- last modified time
    - boolean setLastModified(long time) -- change last modified time

Java IO

- Java File IO is done with Java IO streams.
- Stream is abstraction of data source/sink.
    - Data source -- InputStream or Reader
    - Data sink -- OutputStream or Writer
- Java supports two types of IO streams.
    - Byte streams (binary files) -- byte by byte read/write
    - Character streams (text files) -- char by char read/write
- All these streams are AutoCloseable (so can be used with try-with-resource construct)

Java IO Framework

Java IO Streams
(flow of data)

- Binary/Byte Streams (flow of bytes)
  - OutputStream (to write - sink)
    - void write(byte b);*
    - void write(byte[]);
    - void flush();
    - void close();
  - InputStream (to read - source)
    - int read();*
    - int read(byte[]);
    - void close();
- Character Streams (flow of chars)
  - Writer (to write - sink)
    - void write(char b);*
    - void write(char[]);
    - void flush();
    - void close();
  - Reader (to read - source)
    - int read();*
    - int read(char[]);
    - void close();

OutputStream hierarchy

1. FileOutputStream
   - write files on disk.
2. DataOutputStream
   - primitive types → bytes
3. ObjectOutputStream
   - java object → bytes
4. PrintStream
   - primitive types → text
   - formatted output
5. BufferedOutputStream
   - holds data in memory before writing to underlying stream to improve IO efficiency.

File Copy Program

```
// input src & dest file paths from user.

// create FileInputStream obj - src file.
FileInputStream fin = new FIS(srcPath);
// create FileOutputStream obj - dest file.
FileOutputStream fout = new FOS(destPath);

    // read a byte from in strm
while((b = fin.read()) != -1) {
    // write that byte to out strm
    fout.write(b);
} // repeat above two steps until end of file

// close in & out strm
fout.close();
fin.close();
```

File Edit Selection View Go ...

You are screen sharing Stop Share

day16.md day15.md classwork.md U

C: > Nilesh > onlinecourses > CJ-H-02 > day16 > classwork.md > # Day16 - Classwork > ## demo05 - Annotations

45

Stream characteristics:

1. Immutable
2. Lazily Evaluated
3. No storage
4. Not reusable

Stream operations:

|- Terminal operations
| - Returns non-stream - forEach(), collect(), reduce(), ...
|- Intermediate operations
| - Returns stream
|- Stateless operation - map(), filter(), flatMap(), ...
|- Stateful operation - sorted(), distinct(), ...

main* 0 0 0 Ln 45, Col 1 Spaces: 4 UTF-8 CRLF Markdown

Search 9:06 AM

day16 - demo01/src/com/sunbeam/Program01.java - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program01.java

```
            .map(line -> line.toLowerCase())
            .flatMap(line -> Arrays.stream(line.split("[ ,\\.]"))) // needs a lambda expr: input is one element, and ou
            .filter(word -> !word.isEmpty())
            .distinct()
            .forEach(word -> System.out.println(word));
    }
    */

    public static void main(String[] args) {
        Set<Integer> set = new TreeSet<>();
        Collections.addAll(set, 1, 2, 3, 4, 5, 6);

        Stream<Integer> strm = set.stream();
        strm.forEach(e -> System.out.println(e));

        |
    }
}
```

T reduce(T seed, BinaryOperator<T> op);

```
int res= strm.reduce(0, (a,i)->a + i);
```

Writable | Smart Insert | 47 : 9 : 1493

Search 9:35 AM

day16 - demo03/src/com/sunbeam/Program03.java -

File Edit Source Refactor Navigate Search Project Run Window Help

Mute Start Video Security Participants Chat New Share Pause Share Apps More

You are screen sharing Stop Share

Program03.java

```java
enum Arithmetic {
    EXIT, ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION
}

public class Program03 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter two numbers: ");
        int num1 = sc.nextInt();
        int num2 = sc.nextInt();
        int result;
        //System.out.println("\n0. Exit\n1. Add\n2. Subtract\n3. Multiply\n4. Divide\nEnter choice: ");
        //int choice = sc.nextInt();
        Arithmetic choice = Arithmetic.MULTIPLICATION;
        switch (choice) {
        case ADDITION:
            result = num1 + num2;
            System.out.println("Result: " + result);
            break;
        case SUBTRACTION:
            result = num1 - num2;
            System.out.println("Result: " + result);
            break;
        case MULTIPLICATION:
            result = num1 * num2;
            System.out.println("Result: " + result);
```

```java
class Order {
    int custId;
    int prodId;
    Date orderDate;
    // ...
    String status; // pending, dispatched, paid
    OrderStatus status;
}

enum OrderStatus {
    PENDING, DISPATCHED, PAID
};
```

Writable Smart Insert 20 : 15 : 567

Search 11:41 AM

You are screen sharing
Stop Share


```java
// generated enum code
final class ArithmeticOperations extends Enum {
    public static ArithmeticOperations[] values() {
        return (ArithmeticOperations[])$VALUES.clone();
    }
    public static ArithmeticOperations valueOf(String s) {
        return (ArithmeticOperations)Enum.valueOf(ArithmeticOperations, s);
    }
    private ArithmeticOperations(String name, int ordinal) {
        super(name, ordinal); // invoke sole constructor Enum(String,int);
    }
    public static final ArithmeticOperations ADDITION;
    public static final ArithmeticOperations SUBTRACTION;
    public static final ArithmeticOperations MULIPLICATION;
    public static final ArithmeticOperations DIVISION;
    private static final ArithmeticOperations $VALUES[];
    static {
        ADDITION = new ArithmeticOperations("ADDITION", 0);
        SUBTRACTION = new ArithmeticOperations("SUBTRACTION", 1);
        MULIPLICATION = new ArithmeticOperations("MULIPLICATION", 2);
        DIVISION = new ArithmeticOperations("DIVISION", 3);
        $VALUES = (new ArithmeticOperations[] {
            ADDITION, SUBTRACTION, MULIPLICATION, DIVISION
        });
    }
```

"ADDITION" --> ADDITION object

enum ArithmeticOperations {
ADDITION, SUBTRACTION,..
}

java.lang.Enum class
1. fields: name String, ordinal int.
2. methods:
   a. label() -- returns name string
   b. ordinal() -- returns pos/ord
   c. Enum(label,ordinal) -- ctor
   d. toString() -- returns name.
   e. valueOf() -- String to enum.

JVM architecture

day17 - Spring Tool Suite 4

File Edit Navigate Search Project Run Window Help

Marker interface -- Attach some metadata with the class

- Marks class with some special functionality
- e.g. Cloneable, Serializable, ...
- Limitations: Can be applied to class (not to methods, fields, constructors, args).
  - Limited metadata (no extra info/attributes/details).

Annotations -- To associate additional metadata with the class.

Since Java 5.0

Can be applied to class/interfaces, methods, fields, constructors, method args, local vars, ...

Also has additional attributes/details.

* Retention level: SOURCE, CLASS, or RUNTIME
* Types: Meta-annotations, Annotations
* Pre-defined annotations: @Override, @FunctionalInterface, @SupressWarning, @Deprecated, ...
* Custom/User-defined annotations.

Search 9:59 AM

day17 - Spring Tool Suite 4

File Edit Navigate Search Project Run Window Help

SOURCE Retention

@Override
void method() { ... }

Source.java

→ Java Compiler → ~~@Override~~ → JVM (execution)

To give info/instruction to the Java compiler.

.class

CLASS Retention

@keepName
p.s.v. main(...) { ... }

.java

→ Java Compiler → @KeepName main: → JVM (execution) ~~@KeepName~~

To give info/instructions to special .class based utilities e.g. obfuscator

.class

RUNTIME Retention

@Table(...)
class Book { ... }

.java

→ Java Compiler → @Table ... Book ... → JVM (execution)

@Table
Class

.class

Runtime annotations can be read at runtime using Reflection getAnnotations() and then process.

Search 10:10 AM

day17 - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

> demo01 [CJ-H-02 main]

> demo02 [CJ-H-02 main]

File = Collection of data/info on storage device.

= Data (Contents) + Metadata (Information).

Java File IO

- Deal with File metata -- File system operations -- java.io.File class
- Deal with File data -- File IO operations -- java.io.FileInputStream/FileOutputStream.

demo02

Search

11:39 AM

day17 - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

> demo01 [CJ-H-02 main]

> demo02 [CJ-H-02 main]

java.io.File -- represents a path (of file or directory)

```
File f = new File(path);
```

Methods in File class:

- exists(), isDirectory(), isFile()
- canRead(), canWrite(), canExecute() -- check file/folder permissions
- setReadable(), setWritable(), setExecutable() -- set permissions
- length() -- file info
- list(), listFiles() -- directory listing

demo02

Search

11:42 AM

Core Java

IO Framework

Chaining IO Streams

- Each IO stream object performs a specific task.
  - FileOutputStream -- Write the given bytes into the file (on disk).
  - BufferedOutputStream -- Hold multiple elements in a temporary buffer before flushing it to underlying stream/device. Improves performance.
  - DataOutputStream -- Convert primitive types into sequence of bytes. Inherited from DataOutput interface.
  - ObjectOutputStream -- Convert object into sequence of bytes. Inherited from ObjectOutput interface.
  - PrintStream -- Convert given input into formatted output.
  - Note that input streams does the counterpart of OutputStream class hierarchy.
- Streams can be chained to fulfil application requirements.

Primitive types IO

- DataInputStream & DataOutputStream -- convert primitive types from/to bytes
  - primitive type --> DataOutputStream --> bytes --> FileOutputStream --> file.
    - DataOutput interface provides methods for conversion - writeInt(), writeUTF(), writeDouble(), ...
  - primitive type <-- DataInputStream <-- bytes <-- FileInputStream <-- file.
    - DataInput interface provides methods for conversion - readInt(), readUTF(), readDouble(), ...

DataOutput/DataInput interface

- interface DataOutput
  - writeUTF(String s)
  - writeInt(int i)
  - writeDouble(double d)
  - writeShort(short s)
  - ...
- interface DataInput
  - String readUTF()
  - int readInt()
  - double readDouble()
  - short readShort()
  - ...

Serialization

- ObjectInputStream & ObjectOutputStream -- convert java object from/to bytes

    - Java object --> ObjectOutputStream --> bytes --> FileOutputStream --> file.
        - ObjectOutput interface provides method for conversion - writeObject().
    - Java object <-- ObjectInputStream <-- bytes <-- FileInputStream <-- file.
        - ObjectInput interface provides methods for conversion - readObject().

- Converting state of object into a sequence of bytes is referred as Serialization. The sequence of bytes includes object data as well as metadata.

- Serialized data can be further saved into a file (using FileOutputStream) or sent over the network (Marshalling process).

- Converting (serialized) bytes back to the Java object is referred as Deserialization.

- These bytes may be received from the file (using FileInputStream) or from the network (Unmarshalling process).

ObjectOutput/ObjectInput interface

- interface ObjectOutput extends DataOutput
    - writeObject(obj)
- interface ObjectInput extends DataInput
    - obj = readObject()

Serializable interface

- Object can be serialized only if class is inherited from Serializable interface; otherwise writeObject() throws NotSerializableException.
- Serializable is a marker interface.

transient fields

- writeObject() serialize all non-static fields of the class. If fields are objects, then they are also serialized.
- If any field is intended not to serialize, then it should be marked as "transient".
- The transient and static fields (except serialVersionUID) are not serialized.

serialVersionUID field

- Each serializable class is associated with a version number, called a serialVersionUID.
- It is recommended that programmer should define it as a static final long field (with any access specifier). Any change in class fields expected to modify this serialVersionUID.

```
private static final long serialVersionUID = 1001L;
```

- During deserialization, this number is verified by the runtime to check if right version of the class is loaded in the JVM. If this number mismatched, then InvalidClassException will be thrown.
- If a serializable class does not explicitly declare a serialVersionUID, then the runtime will calculate a default serialVersionUID value for that class (based on various aspects of the class described in the Java(TM) Object Serialization specification).

Buffered streams

- Each write() operation on FileOutputStream will cause data to be written on disk (by OS). Accessing disk frequently will reduce overall application performance. Similar performance problems may occur during network data transfer.
- BufferedOutputStream classes hold data into a in-memory buffer before transferring it to the underlying stream. This will result in better performance.
    - Java object --> ObjectOutputStream --> BufferedOutputStream --> FileOutputStream --> file on disk.
- Data is sent to underlying stream when buffer is full or flush() called explicitly.
- BufferedInputStream provides a buffering while reading the file.
- The buffer size can be provided while creating the respective objects.

PrintStream class

- Produce formatted output (in bytes) and send to underlying stream.
- Formatted output is done using methods print(), println(), and printf().
- System.out and System.err are objects of PrintStream class.

Scanner class

- Added in Java 5 to get the formatted input.
- It is java.util package (not part of java io framework).

```
Scanner sc = new Scanner(inputStream);
// OR
Scanner sc = new Scanner(inputFile);
```

- Helpful to read text files line by line.

Character streams

- Character streams are used to interact with text file.
- Java char takes 2 bytes (unicode), however char stored in disk file may take 1 or more bytes depending on char encoding.
    - https://www.w3.org/International/questions/qa-what-is-encoding
- The character stream does conversion from java char to byte representation and vice-versa (as per char encoding).
- The abstract base classes for the character streams are the Reader and Writer class.
- Writer class -- write operation
    - void close() -- close the stream
    - void flush() -- writes data (in memory) to underlying stream/device.
    - void write(char[] b) -- writes char array to underlying stream/device.
    - void write(int b) -- writes a char to underlying stream/device.
- Writer Sub-classes
    - FileWriter, OutputStreamWriter, PrintWriter, BufferedWriter, etc.
- Reader class -- read operation

    - void close() -- close the stream
    - int read(char[] b) -- reads char array from underlying stream/device
    - int read() -- reads a char from the underlying device/stream. Returns -1
- Reader Sub-classes
    - FileReader, InputStreamReader, BufferedReader, etc.

Java NIO

- Java NIO (New IO) is an alternative IO API for Java.
- Java NIO offers a different IO programming model than the traditional IO APIs.
- Since Java 7.
- Java NIO enables you to do non-blocking (not fully) IO.
- Java NIO consist of the following core components:
    - Channels e.g. FileChannel, ...
    - Buffers e.g. ByteBuffer, ...
    - Selectors
- Java NIO also provides "helper" classes Paths & Files.
    - exists()
    - ...

Paths and Files

- A Java Path instance represents a path in the file system. A path can point to either a file or a directory. A path can be absolute or relative.

```
Path path = Paths.get("c:\\data\\myfile.txt");
```

- Files class (Files) provides several static methods for manipulating files in the file system.

```
static InputStream newInputStream(Path, OpenOption...) throws IOException;
static OutputStream newOutputStream(Path, OpenOption...) throws IOException;
static DirectoryStream<Path> newDirectoryStream(Path) throws IOException;
static Path createFile(Path, attribute.FileAttribute<?>...) throws
IOException;
static Path createDirectory(Path, attribute.FileAttribute<?>...) throws
IOException;
static void delete(Path) throws IOException;
static boolean deleteIfExists(Path) throws IOException;
static Path copy(Path, Path, CopyOption...) throws IOException;
static Path move(Path, Path, CopyOption...) throws IOException;
static boolean isSameFile(Path, Path) throws IOException;
static boolean isHidden(Path) throws IOException;
static boolean isDirectory(Path, LinkOption...);
static boolean isRegularFile(Path, LinkOption...);
static long size(Path) throws IOException;
static boolean exists(Path, LinkOption...);
static boolean isReadable(Path);
static boolean isWritable(Path);
```

```
static boolean isExecutable(Path);
static List<String> readAllLines(Path) throws IOException;
static Stream<String> lines(Path) throws IOException;
```

Channels and Buffers

- All IO in NIO starts with a Channel. A Channel is similar to IO stream. From the Channel data can be read into a Buffer. Data can also be written from a Buffer into a Channel.

NIO Channels

- Java NIO Channels are similar to IO streams with a few differences:
  - You can both read and write to a Channels. Streams are typically one-way (read or write).
  - Channels can be read and written asynchronously (non-blocking).
  - Channels always read to, or write from, a Buffer.
- Channel Examples
  - FileChannel
  - DatagramChannel // UDP protocol
  - SocketChannel, ServerSocketChannel // TCP protocol

NIO Buffers

- A buffer is essentially a block of memory into which you can write data, which you can then later read again. This memory block is wrapped in a NIO Buffer object, which provides a set of methods that makes it easier to work with the memory block.
- Using a Buffer to read and write data typically follows this 4-step process:
  - Write data into the Buffer
  - Call buffer.flip()
  - Read data out of the Buffer
  - Call buffer.clear() or buffer.compact()
- Buffer Examples
  - ByteBuffer
  - CharBuffer
  - DoubleBuffer
  - FloatBuffer
  - IntBuffer
  - LongBuffer
  - ShortBuffer

Channel and Buffer Example

```
RandomAccessFile aFile = new RandomAccessFile("somefile.txt", "rw");
FileChannel inChannel = aFile.getChannel();

ByteBuffer buf = ByteBuffer.allocate(32);

int bytesRead = inChannel.read(buf); // write data into buffer (from channel)
```

```java
while (bytesRead != -1) {
    System.out.println("Read " + bytesRead);
    buf.flip(); // switch buffer from write mode to read mode

    while(buf.hasRemaining()){
        System.out.print((char) buf.get()); // read data from the buffer
    }

    buf.clear(); // clear the buffer
    bytesRead = inChannel.read(buf);
}
aFile.close();
```

RandomAccessFile

- RandomAccessFile class from java.io package.
- Capable of reading and writing into a file (on a storage device).
- Internally maintains file read/write position/cursor.
- Homework: Read docs.

Java NIO vs Java IO

- IO: Stream-oriented
- NIO: Buffer-oriented
- IO: Blocking IO
- NIO: Non-blocking IO

Data Streams

```
Movie m = new Movie(....);
FOS fout = new FOS("path");
DOS dout = new DOS(fout);
dout.writeInt(m.getId());
dout.writeUTF(m.getTitle());
dout.writeDouble(m.getRating());
dout.close();
fout.close();
```

```
Movie m = new Movie();
FIS fin = new FIS("path");
DIS din = new DIS(fin);
m.setId(din.readInt());
m.setTitle(din.readUTF());
m.setRating(din.readDouble());
din.close();
fin.close();
```

Object Streams

```
Movie m=new Movie(....);
FOS fout=new FOS("path");
OOS oout = new OOS(fout);

oout.writeObject(m);

oout.close();
fout.close();
```

```
FIS fin=new FIS("path");
OIS oin=new OIS(fin);

Movie m=(Movie)oin.readObject();

oin.close();
fin.close();
```

Buffered Streams

```
FOS fout = new FOS("path");
BOS bout = new BOS(fout);
DOS dout = new DOS(bout);
dout.writeInt(m.getId());...

dout.close();
bout.close();
fout.close();
```

PrintStream

```
Movie m = new Movie(...);
FOS fout = new FOS("path");
PS out = new PS(fout);
pout.printf("%d,%s,%f\n",
    m.getId(), m.getTitle(), m.getRating());
out.close();
fout.close();
```

Agenda

- JDBC

Java Database Connectivity (JDBC)

- RDBMS understand SQL language only.
- JDBC driver converts Java requests in database understandable form and database response in Java understandable form.
- JDBC drivers are of 4 types
    - Type I - Jdbc Odbc Bridge driver
        - ODBC is standard of connecting to RDBMS (by Microsoft).
        - Needs to create a DSN (data source name) from the control panel.
        - From Java application JDBC Type I driver can communicate with that ODBC driver (DSN).
        - The driver class: sun.jdbc.odbc.JdbcOdbcDriver -- built-in in Java.
        - database url: jdbc:odbc:dsn
        - Advantages:
            - Can be easily connected to any database.
        - Disadvantages:
            - Slower execution (Multiple layers).
            - The ODBC driver needs to be installed on the client machine.
    - Type II - Partial Java/Native driver
        - Partially implemented in Java and partially in C/C++. Java code calls C/C++ methods via JNI.
        - Different driver for different RDBMS. Example: Oracle OCI driver.
        - Advantages:
            - Faster execution
        - Disadvantages:
            - Partially in Java (not truely portable)
            - Different driver for Different RDBMS
    - Type III - Middleware/Network driver
        - Driver communicate with a middleware that in turn talks to RDBMS.
        - Example: WebLogic RMI Driver
        - Advantages:
            - Client coding is easier (most task done by middleware)
        - Disadvantages:
            - Maintaining middleware is costlier
            - Middleware specific to database
    - Type IV
        - Database specific driver written completely in Java.
        - Fully portable.
        - Most commonly used.
        - Example: Oracle thin driver, MySQL Connector/J, ...

MySQL Programming Steps

- step 0: Add JDBC driver into project/classpath.

  - Project Properties -> Java Build Path -> Libraries - Classpath -> Add External Jar -> select mysql driver jar -> Ok
- step 1: Load and register JDBC driver class. These drivers are auto-registered when loaded first time in JVM. This step is optional in Java SE applications from JDBC 4 spec.

```
Class.forName("com.mysql.cj.jdbc.Driver");
    // for Oracle: Use driver class oracle.jdbc.driver.OracleDriver
```

- step 2: Create JDBC connection using helper class DriverManager.

```
// db url = jdbc:dbname://db-server:port/database
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/dbname", "root",
"manager");
    // for Oracle: jdbc:oracle:thin:@localhost:1521:sid
```

- step 3: Create the statement.

```
Statement stmt = con.createStatement();
```

- step 4: Execute the SQL query using the statement and process the result.

```
String sql = "non-select query";
int count = stmt.executeUpdate(sql); // returns number of rows affected
```

  - OR

```
String sql = "select query";
ResultSet rs = stmt.executeQuery(sql);
while(rs.next()) // fetch next row from db (return false when all rows
completed)
{
    x = rs.getInt("col1");      // get first column from the current row
    y = rs.getString("col2");   // get second column from the current row
    z = rs.getDouble("col3");   // get third column from the current row
    // process/print the result
}
rs.close();
```

- step 5: Close statement and connection.

```
stmt.close();
con.close();
```

MySQL Driver Download

- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j/8.4.0

JDBC concepts

java.sql.Driver

- Implemented in JDBC drivers.
  - MySQL: com.mysql.cj.jdbc.Driver
  - Oracle: oracle.jdbc.OracleDriver
  - Postgres: org.postgresql.Driver
- Driver needs to be registered with DriverManager before use.
- When driver class is loaded, it is auto-registered (Class.forName()).
- Driver object is responsible for establishing database "Connection" with its connect() method.
- This method is called from DriverManager.getConnection().

java.sql.Statement

- Represents SQL statement/query.
- To execute the query and collect the result.

```
Statement stmt = con.createStatement();
```

```
ResultSet rs = stmt.executeQuery(selectQuery);
```

```
int count = stmt.executeUpdate(nonSelectQuery);
```

- Since query built using string concatenation, it may cause SQL injection.

java.sql.ResultSet

- ResultSet represents result of SELECT query. The result may have one/more rows and one/more columns.
- Can access only the columns fetched from database in SELECT query (projection).

```
// SELECT id, quote, created_at FROM quotes
ResultSet rs = stmt.executeQuery();
```

```
while(rs.next()) {
    int id = rs.getInt("id");
    String quote = rs.getString("quote");
    Timestamp createdAt = rs.getTimestamp("created_at"); //
java.sql.Timestamp
    // ...
}
```

```
// SELECT id, quote, created_at FROM quotes
ResultSet rs = stmt.executeQuery();
while(rs.next()) {
    int id = rs.getInt(1);
    String quote = rs.getString(2);
    Timestamp createdAt = rs.getTimestamp(3); // java.sql.Timestamp
    // ...
}
```

JDBC Drivers

JDBC Specification is implemented by JDBC Drivers.

There are four types of JDBC Drivers

- Type I – JDBC ODBC Bridge
- Type II – Partial Java Driver
- Type III – Middleware Driver
- Type IV – Pure Java Driver

Search

JDBC Specification

JDBC is a specification by Sun/Oracle to connect with RDBMS.

JDBC Specification includes

- Helper classes
- JDBC Interfaces

JDBC interfaces

- java.sql.Driver interface
  - Makes database connection
- java.sql.Connection interface
  - Represent a session with RDBMS
  - Create SQL statements
- java.sql.Statement interface
  - Represent SQL statement
  - PreparedStatement and CallableStatement
- java.sql.ResultSet interface
  - Process result of SELECT queries

day18 - demo01/src/com/sunbeam/Program01.java - Spring Too

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Package Exp... | Program01.java | Movie.java

- demo01 [CJ-H-02-
  - JRE System Library
  - src
    - com.sunbear
      - Movie.java
      - Program01.j

```java
            e.printStackTrace();
        }
    }

    private static void readMovies() {
        List<Movie> list = new ArrayList<>();
        try(FileInputStream fin = new FileInputStream("movies.db")) {
            try(DataInputStream din = new DataInputStream(fin)) {
                while(true) {
                    Movie m = new Movie();
                    m.setId( din.readInt() );
                    m.setTitle( din.readUTF() );
                    m.setRating( din.readDouble() );
                    list.add(m);
                }
            } // din.close();
        } // fin.close();
        catch (EOFException e) {
            // do nothing
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("Movie List: ");
        list.forEach(m -> System.out.println(m));
```

EOFException

Writable | Smart Insert | 51 : 30 : 1494

Search | 10:11 AM

day18 - demo02/src/com/sunbeam/Movie.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Exp... | Program01.java | Movie.java | Program02.java | Movie.java

```java
import java.io.Serializable;

public class Movie implements Serializable {
    private static final long serialVersionUID = 1L; // represents object structure/schema version
    private int id;
    private String title;
    private double rating;
    transient private String director; // transient fields are not serialized and deserialized
    static String theatre = "PVR";
    public Movie() {
    }
    public Movie(int id, String title, double rating) {
        this.id = id;
        this.title = title;
        this.rating = rating;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    }
    public void setTitle(String title) {
```

Writable | Smart Insert | 17 : 30 : 532

Search 10:51 AM

day18 - Spring Tool Suite 4

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Package Exp...

- demo01 [CJ-H-02
  - JRE System Library
  - src
    - com.sunbear
      - Movie.java
      - Program01.j
- demo02 [CJ-H-02
  - JRE System Library
  - src
    - com.sunbear
      - Movie.java
      - Program02.j
- demo03 [CJ-H-02
  - JRE System Library
  - src
    - com.sunbear
      - Movie.java
      - Program03.j

PrintStream class:

```
    println(String str);
    printf(String fmt, ...);
```

PrintStream class objects -- System.out, System.err --> console/terminal

PrintStream class object --> FileOutputStream object --> file on disk

Search

11:55 AM

Java IO Framework

'A' --> 65 (ASCII)

ASCII -- English (1-byte)
UTF-16-- Main lang (2-by)

H L

=hex 2665.

LE	65	26
BE	26	65

Java IO Streams
(flow of data)

Char streams are used to handle char encoding.
Char encoding -- How chars are stored in file/disk.e.g. ASCII, UNICODE (UTF-8, UTF-16, UTF16-BE, UTF16-LE, UTF32, ...). depending on char set to support.

Binary/Byte Streams
(flow of bytes)

Character Streams
(flow of chars)

OutputStream
(to write - sink)

```
void write(byte b);*
void write(byte[]);
void flush();
void close();
```

InputStream
(to read - source)

```
int read();*
int read(byte[]);
void close();
```

Writer
(to write - sink)

```
void write(char b);*
void write(char[]);
void flush();
void close();
```

Reader
(to read - source)

```
int read();*
int read(char[]);
void close();
```

day18 - demo02/src/com/sunbeam/Program02.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
- demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE
  - src
    - com.sunbeam
      - Movie.java
      - Program02.java
- demo03 [CJ-H-02 main]
- demo04 [CJ-H-02 main]
- demo05 [CJ-H-02 main]
- demo06 [CJ-H-02 main]
  - src
    - com.sunbeam
  - JRE System Library [jre]

Program06.java | Program02.java

```
17      }
18
19      private static void writeMovies() {
20          List<Movie> list = new ArrayList<>();
21          list.add(new Movie(1, "Star Wars", 7.5));
22          list.add(new Movie(2, "Godfather", 8.0));
23          list.add(new Movie(3, "Hidden Figures", 7.0));
24          list.add(new Movie(4, "Bruce Almighty", 6.5));
25          list.add(new Movie(5, "Forest Gump", 8.5));
26
27          try(FileOutputStream fout = new FileOutputStream("movies.db")) {
28              try(ObjectOutputStream oout = new ObjectOutputStream(fout)) {
29                  oout.writeObject(list);
30              } // oout.close();
31              System.out.println("Movies Saved.");
32          } // fout.close();
33          catch (Exception e) {
34              e.printStackTrace();
35          }
36      }
37
38      private static void readMovies() {
39          List<Movie> list = new ArrayList<>();
40          try(FileInputStream fin = new FileInputStream("movies.db")) {
41              try(ObjectInputStream oin = new ObjectInputStream(fin)) {
42                  list = (List<Movie>) oin.readObject();
```

OOS → bytes → FOS → file

|

metadata

+ data

ArrayList

+ Movies

Writable | Smart Insert | 29 : 40 [23]

Search 1:18 PM

You are screen sharing Stop Share

Writer hierarchy

```
FileWriter wr = new FileWriter(path);  ---------->  wr.write(ch);
```

same as --

```
FileOutputStream fout = new FileOutputStream(path);
OutputStreamWriter wr = new OutputStreamWriter(fout);--> wr.write(ch);
```

FileWriter = OutputStreamWriter + FileOutputStream

Core Java

PreparedStatement

- PreparedStatement represents parameterized queries.

```
String sql = "SELECT * FROM students WHERE name=?";
PreparedStatement stmt = con.prepareStatement(sql);
System.out.print("Enter name to find: ");
String name = sc.next();
stmt.setString(1, name);
ResultSet rs = stmt.executeQuery();
while(rs.next()) {
    int roll = rs.getInt("roll");
    String name = rs.getString("name");
    double marks = rs.getDouble("marks");
    System.out.printf("%d, %s, %.2f\n", roll, name, marks);
}
```

- The same PreparedStatement can be used for executing multiple queries. There is no syntax checking repeated. This improves the performance.

MySQL Programming steps -- PreparedStatement

1. Add JDBC driver into project/classpath.
    - Java project -> Properties -> Java Build Path -> Libraries -> Add External Jars -> select MySQL JDBC driver jar -> Apply and Close.
2. Load and register driver class.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

3. Create database connection.

```
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/dbname", "dbuser",
"dbpassword");
```

4. Create PreparedStatement with (paramterized) SQL query.

```
String sql = "sql query with ?";
PreparedStatement stmt = con.prepareStatement(sql);
```

5. Set param values, execute the query and process the result.

```
stmt.setInt(1, val1); // set 1st param ? value
stmt.setString(2, val2); // set 2nd param ? value
```

```
// for non-SELECT queries
int count = stmt.executeUpdate();
```

```
// for SELECT queries
ResultSet rs = stmt.executeQuery();
while(rs.next()) {
    int val1 = rs.getInt("col1");
    String val2 = rs.getString("col2");
    // ...
}
rs.close();
```

6. Close statement and connection.

```
stmt.close();
con.close();
```

java.sql.PreparedStatement

- Inherited from java.sql.Statement.
- Represents parameterized SQL statement/query.
- The query parameters (?) should be set before executing the query.
- Same query can be executed multiple times, with different parameter values.
- This speed up execution, because query syntax checking is done only once.

```
PreparedStatement stmt = con.prepareStatement(query);
```

```
stmt.setInt(1, intValue);
stmt.setString(2, stringValue);
stmt.setDouble(3, doubleValue);
stmt.setDate(4, dateObject); // java.sql.Date
stmt.setTimestamp(5, timestampObject); // java.sql.Timestamp
```

```
ResultSet rs = stmt.executeQuery();
// OR
int count = stmt.executeUpdate();
```

Call Stored Procedure using JDBC (without OUT parameters)

- Stored Procedure - Change price of given book id.

```
DELIMITER //

CREATE PROCEDURE sp_updateprice(IN p_id INT, IN p_price DOUBLE)
BEGIN
    UPDATE books SET price=p_price WHERE id=p_id;
END;
//

DELIMITER ;
```

```
CALL sp_updateprice(22, 543.21);
```

- JDBC use CallableStatement interface to invoke the stored procedures.
- CallableStatement interface is extended from PreparedStatement interface.
- Steps to call Stored procedure are same as PreparedStatement.
    - Create connection.
    - Create CallableStatement using con.prepareCall("CALL ...").
    - Set IN parameters using stmt.setXYZ(...);
    - Execute the procedure using stmt.executeQuery() or stmt.executeUpdate().
    - Close statement & connection.
- To invoke stored procedure, in general stmt.execute() is called. This method returns true, if it is returning ResultSet (i.e. multi-row result). Otherwise it returns false, if it is returning update/affected rows count.

```
boolean isResultSet = stmt.execute();
if(isResultSet) {
    ResultSet rs = stmt.getResultSet();
    // process the ResultSet
}
else {
    int count = stmt.getUpdateCount();
    // process the count
}
```

Call Stored Procedure using JDBC (with OUT parameters)

- Stored Procedure - Get title and price of given book id.

```
DELIMITER //

CREATE PROCEDURE sp_gettitleprice(IN p_id INT, OUT p_name CHAR(40), OUT
p_price DOUBLE)
BEGIN
    SELECT name INTO p_name FROM books WHERE id=p_id;
    SELECT price INTO p_price FROM books WHERE id=p_id;
END;
//

DELIMITER ;
```

```
CALL sp_gettitleprice(22, @p_name, @p_price);

SELECT @p_name, @p_price;
```

- Steps to call Stored procedure with out params.
    - Create connection.
    - Create CallableStatement using con.prepareCall("CALL ...").
    - Set IN parameters using stmt.setXYZ(...) and register out parameters using stmt.registerOutParam(...).
    - Execute the procedure using stmt.execute().
    - Get values of out params using stmt.getXYZ(paramNumber).
    - Close statement & connection.

Transaction Management

- RDBMS Transactions
    - Transaction is set of DML operations to be executed as a single unit. Either all queries in tx should be successful or all should be discarded.
    - The transactions must be atomic. They should never be partial.

```
CREATE TABLE accounts(id INT, type CHAR(30), balance DOUBLE);
INSERT INTO accounts VALUES (1, 'Saving', 30000.00);
INSERT INTO accounts VALUES (2, 'Saving', 2000.00);
INSERT INTO accounts VALUES (3, 'Saving', 10000.00);

SELECT * FROM accounts;

START TRANSACTION;
```

```sql
--SET @@autocommit=0;

UPDATE accounts SET balance=balance-4000 WHERE id=1;
UPDATE accounts SET balance=balance+4000 WHERE id=2;

SELECT * FROM accounts;

COMMIT;
-- OR
ROLLBACK;
```

- JDBC transactions (Logical code)

```java
try(Connection con = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD)) {
    con.setAutoCommit(false); // start transaction
    String sql = "UPDATE accounts SET balance=balance+? WHERE id=?";
    try(PreparedStatement stmt = con.prepareStatement(sql)) {
        stmt.setDouble(1, -3000.0); // amount=3000.0
        stmt.setInt(2, 1); // accid = 1
        cnt1 = stmt.executeUpdate();
        stmt.setDouble(1, +3000.0); // amount=3000.0
        stmt.setInt(2, 2); // accid = 2
        cnt2 = stmt.executeUpdate();
        if(cnt1 == 0 || cnt2 == 0)
            throw new RuntimeException("Account Not Found");
    }
    con.commit(); // commit transaction
}
catch(Exception e) {
    e.printStackTrace();
    con.rollback(); // rollback transaction
}
```

ResultSet

- ResultSet types
  - TYPE_FORWARD_ONLY -- default type
    - next() -- fetch the next row from the db and return true. If no row is available, return false.

```java
while(rs.next()) {
    // ...
}
```

  - TYPE_SCROLL_INSENSITIVE
    - next() -- fetch the next row from the db and return true. If no row is available, return false.

        - previous() -- fetch the previous row from the db and return true. If no row is available, return false.
        - absolute(rownum) -- fetch the row with given row number and return true. If no row is available (of that number), return false.
        - relative(rownum) -- fetch the row of next rownum from current position and return true. If no row is available (of that number), return false.
        - first(), last() -- fetch the first/last row from db.
        - beforeFirst(), afterLast() -- set ResultSet to respective positions.
        - INSENSITIVE -- After taking ResultSet if any changes are done in database, those will NOT be available/accessible using ResultSet object. Such ResultSet is INSENSITIVE to the changes (done externally).
    - TYPE_SCROLL_SENSITIVE
        - SCROLL -- same as above.
        - SENSITIVE -- After taking ResultSet if any changes are done in database, those will be available/accessible using ResultSet object. Such ResultSet is SENSITIVE to the changes (done externally).
- ResultSet concurrency
    - CONCUR_READ_ONLY -- Using this ResultSet one can only read from db (not DML operations). This is default concurrency.
    - CONCUR_UPDATABLE -- Using this ResultSet one can read from db as well as perform INSERT, UPDATE and DELETE operations on database.

```
String sql = "SELECT roll, name, marks FROM students";
stmt = con.prepareStatement(sql, ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
rs = stmt.executeQuery();
```

```
rs.absolute(2); // moves the cursor to the 2nd row of rs
rs.updateString("name", "Bill"); // updates the 'name' column of row 2
to be Bill
rs.updateDouble("marks", 76.32); // updates the 'marks' column of row 2
to be 76.32
rs.updateRow(); // updates the row in the database
```

```
rs.moveToInsertRow(); // moves cursor to the insert row -- is a blank
row
rs.updateInt(1, 9); // updates the 1st column (roll) to be 9
rs.updateString(2, "AINSWORTH"); // updates the 2nd column (name) of to
be AINSWORTH
rs.updateDouble(3, 76.23); // updates the 3rd column (marks) to true
76.23
rs.insertRow(); // inserts the row in the database
rs.moveToCurrentRow();
```

```
rs.absolute(2); // moves the cursor to the 2nd row of rs
rs.deleteRow(); // deletes the current row from the db
```

###Read Uncommitted 1.This is the lowest isolation level. 2.A transaction can read uncommitted changes made by other transactions. 3.This means it can see data that other transactions have modified but not yet committed. 4.This level allows for dirty reads, where a transaction reads data that might later be rolled back. 5.Typically used when performance is the highest priority, and data consistency is less critical (e.g., logging systems).

###Read Committed(Non-repeatable Reads) 1.A transaction can only read committed data (i.e., data that has been committed by other transactions). 2.Dirty reads are prevented, but it can still encounter non-repeatable reads. 3.A non-repeatable read happens when the value of a record is changed by another transaction after the current transaction has read it.

###Repeatable Read(Phantom Reads) 1.A transaction can read data repeatedly, and the data it reads will not change during the transaction, even if other transactions modify it. 2.This level prevents dirty reads and non-repeatable reads. 3.However, phantom reads are still possible (new rows might appear or disappear between reads in the same transaction).

###Serializable 1.This is the highest isolation level. 2.It ensures complete isolation by serializing transactions — meaning they are executed as if they were happening one after the other, with no overlapping. 3.It prevents dirty reads, non-repeatable reads, and phantom reads.

JDBC

* Specification to connect any RDBMS from Java appln.

JDBC converts Java req to db understandable form and DB response to Java understandable form.

DB	Java
① Table	① class
② Row/Column	② objects
③ SQL	

JDBC specs given as set of interfaces & helper classes:

java.sql

* interfaces:
  1. Driver
  2. Connection
  3. Statement
     - ↑ PreparedStatement
       - ↑ CallableStatement
  4. ResultSet

* classes:
  1. DriverManager
  2. Date, Blob, ⋮

JDBC driver → set of classes implementing JDBC interfaces.

(I) Type I driver
JDBC ⇄ ODBC

(II) Type II Driver

(III) Type III driver

(IV) Type IV driver

JDBC programming steps

① add jdbc driver jar into project class path.
Project properties → Java Build Path → Libraries - Classpath → Add external jar + select jdbc driver jar (downloaded) + OK.

② load & register jdbc driver class.
Class.forName("pkg.DriverClassName");

③ create jdbc connection (using DriverManager).
```
url = "jdbc:dbname: ...";
con = DriverManager.getConnection(url, user, passwd);
```

④ create jdbc statement.
```
stmt = con.createStatement();
```

⑤ execute sql query (using stmt) & process result.
```
cnt = stmt.executeUpdate("non-select sql");
```

```
rs = stmt.executeQuery("select sql");
while (rs.next()) {
    val1 = rs.getInt("col1");
    val2 = rs.getString("col2");
    val3 = rs.getDouble("col3");
    val4 = rs.getDate("col4");
    ...
}
rs.close();
```

⑥ close stmt & connection.
```
stmt.close();
con.close();
```

if user input is concatenated to sql query, the malicious content by user may cause unexpected results / damage → SQL Injection

Avoid using Statement (i.e. sql queries with string concat).

JDBC programming steps

① add jdbc driver jar into project class path.
Project properties → Java Build Path → Libraries - Classpath → Add external jar + select jdbc driver jar (downloaded) + OK.

② load & register jdbc driver class.
```
Class.forName("pkg.DriverClassName");
```

③ create jdbc connection (using DriverManager).
```
url = "jdbc:dbname: ...";
con = DriverManager.getConnection(url, user, passwd);
```

④ create jdbc statement.
```
sql = "select or non select sql with params ?"; //parameterized query
stmt = con.prepareStatement(sql);
```

⑤ execute sql query (using stmt) & process result.
```
stmt.setInt(1, val1);
stmt.setString(2, val2);
stmt.setDouble(3, val3);
cnt = stmt.executeUpdate();
```
set value of each param (?).

```
rs = stmt.executeQuery();
while (rs.next()) {
    val1 = rs.getInt("col1");
    val2 = rs.getString("col2");
    ...
}
rs.close();
```

⑥ close stmt & connection.
```
stmt.close();
con.close();
```

Core Java

Process vs Threads

Program

- Program is set of instructions given to the computer.
- Executable file is a program.
- Executable file contains text, data, rodata, symbol table, exe header.

Process

- Process is program in execution.
- Program (executable file) is loaded in RAM (from disk) for execution. Also OS keep information required for execution of the program in a struct called PCB (Process Control Block).
- Process contains text, data, rodata, stack, and heap section.

Thread

- Threads are used to do multiple tasks concurrently within a single process.
- Thread is a lightweight process.
- When a new thread is created, a new TCB is created along with a new stack. Remaining sections are shared with parent process.

Process vs Thread

- Process is a container that holds resources required for execution and thread is unit of execution/scheduling.
- Each process have one thread created by default -- called as main thread.

Process creation (Java)

- In Java, process can be created using Runtime object.
- Runtime object holds information of current runtime environment that includes number of processors, JVM memory usage, etc.
- Current runtime can be accessed using static getRuntime() method.

```
Runtime rt = Runtime.getRuntime();
```

- The process is created using exec() method, which returns the Process object. This object represents the OS process and its waitFor() method wait for the process termination (and returns exit status).

```
String[] args = { "/path/of/executable", "cmd-line arg1", ... };
Process p = rt.exec(args);
int exitStatus = p.waitFor();
```

Multi-threading (Java)

- Java applications are always multi-threaded.
- When any java application is executed, JVM creates (at least) two threads.
  - main thread -- executes the application main()
  - GC thread -- does garbage collection (release unreferenced objects)
- Programmer may create additional threads, if required.

Thread creation

- To create a thread
  - step 1: Implement a thread function (task to be done by the thread)
  - step 2: Create a thread (with above function)
- Method 1: extends Thread

```java
class MyThread extends Thread {
    @Override
    public void run() {
        // task to be done by the thread
    }
}
```

```java
MyThread th = new MyThread();
th.start();
```

- Method 2: implements Runnable

```java
class MyRunnable implements Runnable {
    @Override
    public void run() {
        // task to be done by the thread
    }
}
```

```java
MyRunnable runnable = new MyRunnable();
Thread th = new Thread(runnable);
th.start();
```

- Java doesn't support multiple inheritance. If your class is already inherited from a super class, you cannot extend it from Thread class. Prefer Runnable in this case; otherwise you may choose any method.

```java
// In Java GUI application is inherited from Frame class.
// to create run() in the same class, you must use Runnable
class MyGuiApplication extends Frame implements Runnable {
    // ...
    public void run() {
        // ...
    }
    // ...
}
```

start() vs run()

- run():
  - Programmer implemented code to be executed by the thread.
- start():
  - Pre-defined method in Thread class.
  - When called, the thread object is submitted to the (JVM/OS) scheduler. Then scheduler select the thread for execution and thread executes its run() method.

Thread methods

- static Thread currentThread()
  - Returns a reference to the currently executing thread object.
- static void sleep(long millis)
  - Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.
- static void yield()
  - A hint to the scheduler that the current thread is willing to yield its current use of a processor.
- Thread.State getState()
  - Returns the state of this thread.
  - State can be NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING, TERMINATED
- void run()
  - If this thread was constructed using a separate Runnable run object, then that Runnable object's run method is called. If thread class extends from Thread class, this method should be overridden. The default implementation is empty.
- void start()
  - Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.

- void join()
  - Waits for this thread to die/complete.
- boolean isAlive()
  - Tests if this thread is alive.
- void setDaemon(boolean daemon);
  - Marks this thread as either a daemon thread (true) or a user thread (false).
- boolean isDaemon()
  - Tests if this thread is a daemon thread.
- long getId()
  - Returns the identifier of this Thread.
- void setName(String name)
  - Changes the name of this thread to be equal to the argument name.
- String getName()
  - Returns this thread's name.
- void setPriority(int newPriority)
  - Changes the priority of this thread.
  - In Java thread priority can be 1 to 10.
  - May use predefined constants MIN_PRIORITY(1), NORM_PRIORITY(5), MAX_PRIORITY(10).
- int getPriority()
  - Returns this thread's priority.
- ThreadGroup getThreadGroup()
  - Returns the thread group to which this thread belongs.
- void interrupt()
  - Interrupts this thread -- will raise InterruptedException in the thread.
- boolean isInterrupted()
  - Tests whether this thread has been interrupted.

Daemon threads

- By default all threads are non-daemon threads (including main thread).
- We can make a thread as daemon by calling its setDaemon(true) method -- before starting the thread.

- Daemon threads are also called as background threads and they support/help the non-daemon threads.
- When all non-daemon threads are terminated, the Daemon threads get automatically terminated.

Thread life cycle

- Thread.State state = th.getState();
- NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING, TERMINATED
  - NEW: New thread object created (not yet started its execution).
  - RUNNABLE: Thread is running on CPU or ready for execution. Scheduler picks ready thread and dispatch it on CPU.
  - BLOCKED: Thread is waiting for lock to be released. Thread blocks due to synchronized block/method.
  - WAITING: Thread is waiting for the notification. Waiting thread release the acquired lock.
  - TIMED_WAITING: Thread is waiting for the notification or timeout duration. Waiting thread release the acquired lock.
  - TERMINATED: Thread terminates when run() method is completed, stopped explicitly using stop(), or an exception is raised while executing run().

Synchronization

- When multiple threads try to access same resource at the same time, it is called as Race condition.
- Example: Same bank account undergo deposit() and withdraw() operations simultaneously.
- It may yield in unexpected/undesired results.
- This problem can be solved by Synchronization.
- The synchronized keyword in Java provides thread-safe access.
- Java synchronization internally use the Monitor object associated with any object. It provides lock/unlock mechanism.
- "synchronized" can be used for block or method.
- It acquires lock on associated object at the start of block/method and release at the end. If lock is already acquired by other thread, the current thread is blocked (until lock is released by the locking thread).
- "synchronized" non-static method acquires lock on the current object i.e. "this". Example:

```java
class Account {
    // ...
    public synchronized void deposit(double amount) {
        double newBalance = this.balance + amount;
        this.balance = newBalance;
    }
    public synchronized void withdraw(double amount) {
        double newBalance = this.balance - amount;
        this.balance = newBalance;
    }
}
```

- "synchronized" static method acquires lock on metadata object of the class i.e. MyClass.class. Example:

```java
class MyClass {
    private static int field = 0;
    // called by incThread
    public synchronized static void incMethod() {
        field++;
    }
    // called by decThread
    public synchronized static void decMethod() {
        field--;
    }
}
```

- "synchronized" block acquires lock on the given object.

```java
// assuming that no method in Account class is synchronized.

// thread1
synchronized(acc) {
    acc.deposit(1000.0);
}

// thread2
synchronized(acc) {
    acc.withdraw(1000.0);
}
```

- Alternatively lock can be acquired using RentrantLock since Java 5.0. Example code:

```java
class Example {
    private final ReentrantLock rl = new ReentrantLock();
    public void method() {
        rl.lock();
        try {
            // ...
        }
        finally {
            rl.unlock();
        }
    }
}
```

- Synchronized collections
    - Synchronized collections (e.g. Vector, Hashtable, ...) use synchronized keyword (block/method) to handle race conditions.

Process vs Thread

OS keep info about process execution. → PCB

PCB TCB TCB

process is program under execution.

Thread is a light-weight process.
Threads are used to execute multiple tasks concurrently within a process.

Process is like a container that holds resources required for execution; while thread is unit of execution/scheduling.

Thread creation

extends Thread

```
class MyThread extends Thread {
    :
    @override
    void run() {
        //code
    }
}

in main(),
    MyThread t1 = new MyThread();
    t1.start();
```

If a java class is already inherited from a class, it cannot be inherited from Thread class (multiple class inherit not allowed)

implements Runnable

```
class MyRunnable implements Runnable {
    @override
    void run() {
        //code
    }
}

in main(),
    MyRunnable mr = new MyRunnable();
    Thread t2 = new Thread(mr);
    t2.start();
```

Even if class is inherited from other class, implementing Runnable is possible. Since Runnable is functional interface, we can use lambda expression.

Thread life cycle

Race condition

when multiple threads access same resource at the same time, then race condn occurs → may lead to unexpected/undesired results.

Synchronization

We need to implement synchronization to avoid race condition.

In Java, each object is associated with a sync object called as "monitor". Monitor has two states i.e. locked (busy) and unlocked (available).

Only one thread can lock the monitor and only that thread can unlock it later.

Java has synchronized keyword to lock/unlock the monitor – ① synchronized block ② synchronized method.

RDBMS Transactions

Transaction is set of DML statements that can be executed as single unit.

RDBMS transactions have ACID properties.

- Atomic: Either all statements in the transaction succeed or all are discarded.
- Consistency: Database should be in valid state (integrity) after transaction.
- Isolation: Multiple transactions can execute concurrently without affecting others.
- Durability: All changes must be saved permanently at the end of transaction.

```
mysql> START TRANSACTION;
mysql> -- dml query 1
mysql> -- dml query 2
mysql> COMMIT;
OR     ROLLBACK;
```

JDBC Transactions

In JDBC transaction management is done with Connection interface methods.

- setAutoCommit()
- commit()
- rollback()

```
try {
        con.setAutoCommit(false);
        // create & execute dml stmts
        con.commit();
} catch(Exception e) {
        con.rollback();
}
```

Savepoints

In RDBMS, savepoint represent state of database within a transaction.

Savepoint is useful to restore state within transaction without discarding whole transaction.

- START TRANSACTION;
- dml statements (A)
- SAVEPOINT sa1;
- dml statements (B)
- ROLLABCK TO sa1; -- discards B
- dml statements (C)
- COMMIT; -- commits A & C

In JDBC, savepoints can be handled using Connection interface methods.

- sa = con.setSavepoint();
- con.rollback(sa);

Stored Procedure

Stored Procedure is a set of SQL queries compiled and stored in RDBMS.

Stored Procedure improve the performance, as they are compiled only once while creation.

Stored procedures can take parameters: IN, OUT and INOUT.

```
CREATE PROCEDURE sp_getstudents(IN p_marks DOUBLE)
BEGIN
        SELECT * FROM students WHERE marks > p_marks;
END;
CALL sp_getstudents(80.0);

CREATE PROCEDURE sp_getmarks(IN p_roll INT, OUT p_marks DOUBLE)
BEGIN
        SELECT marks INTO p_marks FROM students WHERE roll = p_roll;
END;
CALL sp_getmarks(1, @m);
SELECT @m;
```

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE sp_getstudents(IN p_marks DOUBLE)
    -> BEGIN
    -> SELECT * FROM students WHERE marks > p_marks;
    -> END;
    -> $$
Query OK, 0 rows affected (0.07 sec)

mysql> DELIMITER ;
mysql> CALL sp_getstudents(98.0);
+------+--------+-------+
| roll | name   | marks |
+------+--------+-------+
|    1 | Myia   |  98.2 |
|    3 | Dennis |    99 |
+------+--------+-------+
2 rows in set (0.02 sec)

Query OK, 0 rows affected (0.02 sec)
```

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE sp_getmarks(IN p_roll INT, OUT p_marks DOUBLE)
    -> BEGIN
    -> SELECT marks INTO p_marks FROM students WHERE roll = p_roll;
    -> END;
    -> $$
Query OK, 0 rows affected (0.03 sec)

mysql> DELIMITER ;
mysql> CALL sp_getmarks(1, @m);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT @m;
+------+
| @m   |
+------+
| 98.2 |
+------+
1 row in set (0.00 sec)
```

day19 - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

OS Functions

==========

1. Process mgmt
2. Memory mgmt
3. File and IO mgmt
4. CPU scheduling
5. Hardward abstraction
6. User interface (GUI)

JVM is dependent on OS for certain functionalities

==========

1. IPC - networking (Sockets)
2. Process and Thread creation/execution
3. File IO
4. GUI programming (AWT)

Java is not fully "Platform Independent".

Search 10:20 AM

day19 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
// thread creation - method 1
    // step1: create a new class inherited from thread class and override its run() method.
    // step2: create object of that thread class and call its start() method.
class MyThread extends Thread {
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("  My: " + i);
            delay(1000);
        }
    }
}
MyThread th1 = new MyThread();
th1.start();
// thread creation - method 2
    // step1: create a new class inherited from Runnable interface and implement its run() method.
    // step2: create object of Thread class with the object of above Runnable class and call thread's start() method
class YourRunnable implements Runnable {
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("Your: " + i);
            delay(1000);
        }
    }
}
Thread th2 = new Thread(new YourRunnable());
th2.start();
for (int i = 1; i <= 10; i++) {
    System.out.println("Main: " + i);
    delay(1000);
}
```

th1

th2

main

Writable | Smart Insert | 38 : 21 : 1167

Search 10:58 AM

day19 - demo04/src/com/sunbeam/Program04.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program04.java

```java
        // thread creation - method 1
            // step1: create a new class inherited from thread class and override its run() method.
            // step2: create object of that thread class and call its start() method.
        class MyThread extends Thread {
            public void run() {
                for (int i = 1; i <= 10; i++) {
                    System.out.println("  My: " + i);
                    delay(1000);
                }
            }
        }
        MyThread th1 = new MyThread();
        th1.start();
        // thread creation - method 2
            // step1: create a new class inherited from Runnable interface and implement its run() method.
            // step2: create object of Thread class with the object of above Runnable class and call thread's star
        class YourRunnable implements Runnable {
            public void run() {
                for (int i = 1; i <= 10; i++) {
                    System.out.println("Your: " + i);
                    delay(1000);
                }
            }
        }
        Thread th2 = new Thread(new YourRunnable());
        th2.start();
        for (int i = 1; i <= 10; i++) {
            System.out.println("Main: " + i);
            delay(1000);
        }
```

Pro... Jav... Dec... Co...

<terminated> Program04 [Java Application

```
  My: 1
Your: 1
Main: 1
  My: 2
Main: 2
Your: 2
  My: 3
Main: 3
Your: 3
  My: 4
Main: 4
Your: 4
  My: 5
Main: 5
Your: 5
  My: 6
Main: 6
Your: 6
  My: 7
Main: 7
Your: 7
  My: 8
Main: 8
Your: 8
  My: 9
Main: 9
```

Search 11:01 AM

day19 - demo06/src/com/sunbeam/Program06.java - Spring Too

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program06.java

```
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        class PrintTable extends Thread {
            private int num;
            public PrintTable(int num) {
                this.num = num;
            }
            @Override
            public void run() {
                for (int i = 1; i <= 10; i++) {
                    System.out.printf("%d * %d = %d\n", num, i, num * i);
                    delay(1000);
                }
            }
        }

        PrintTable th1 = new PrintTable(2);
        th1.start();
        PrintTable th2 = new PrintTable(4);
        th2.start();

        System.out.println("Bye!");
    } when main() exit...
```

JVM architecture:
- Memory areas
  - Stack area
  - Program Counter
  - Heap area

<terminated> Program06 [Java Application

```
Bye!
4 * 1 = 4
2 * 1 = 2
2 * 2 = 4
4 * 2 = 8
2 * 3 = 6
4 * 3 = 12
2 * 4 = 8
4 * 4 = 16
2 * 5 = 10
4 * 5 = 20
2 * 6 = 12
4 * 6 = 24
2 * 7 = 14
4 * 7 = 28
2 * 8 = 16
4 * 8 = 32
2 * 9 = 18
4 * 9 = 36
2 * 10 = 20
4 * 10 = 40
```

Search 11:54 AM

day19 - demo06/src/com/sunbeam/Program06.java - Spring Too

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program06.java

```java
    public static void main(String[] args) {
        class PrintTable extends Thread {
            private int num;
            public PrintTable(int num) {
                this.num = num;
            }
            @Override
            public void run() {
                for (int i = 1; i <= 10; i++) {
                    System.out.printf("%d * %d = %d\n", num, i, num * i);
                    delay(1000);
                }
            }
        }

        PrintTable th1 = new PrintTable(2);
        th1.start();
        PrintTable th2 = new PrintTable(4);
        th2.start();

        try {
            th1.join(); // calling thread i.e. main will wait for completion given thread i.e. th1
            th2.join(); // calling thread i.e. main will wait for completion given thread i.e. th2
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Bye!");
    }
}
```

Pro... Jav... Dec... Co...

<terminated> Program06 [Java Application

```
2 * 1 = 2
4 * 1 = 4
2 * 2 = 4
4 * 2 = 8
2 * 3 = 6
4 * 3 = 12
2 * 4 = 8
4 * 4 = 16
2 * 5 = 10
4 * 5 = 20
2 * 6 = 12
4 * 6 = 24
2 * 7 = 14
4 * 7 = 28
2 * 8 = 16
4 * 8 = 32
2 * 9 = 18
4 * 9 = 36
2 * 10 = 20
4 * 10 = 40
Bye!
```

Search 12:05 PM

day19 - demo06/src/com/sunbeam/Program06.java - Spring Too

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program06.java

```java
    public static void main(String[] args) {
        class PrintTable extends Thread {
            private int num;
            public PrintTable(int num) {
                this.num = num;
            }
            @Override
            public void run() {
                for (int i = 1; i <= 10; i++) {
                    System.out.printf("%d * %d = %d\n", num, i, num * i);
                    delay(1000);
                }
            }
        }
main thread
        PrintTable th1 = new PrintTable(2);
        th1.start();
        PrintTable th2 = new PrintTable(4);
        th2.start();

        try {
            th1.join(); // calling thread i.e. main will wait for completion given thread i.e. th1
            th2.join(); // calling thread i.e. main will wait for completion given thread i.e. th2
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

Writable | Smart Insert | 32 : 1 : 651

Search | 12:08 PM

day19 - demo06/src/com/sunbeam/Program06.java - Spring Too

You are screen sharing | Stop Share

File Edit Source Refactor Navigate Search Project Run Window Help

Program06.java

```java
    public static void main(String[] args) {
        class PrintTable extends Thread {
            private int num;
            public PrintTable(int num) {
                this.num = num;
            }
            @Override
            public void run() {
                for (int i = 1; i <= 10; i++) {
                    System.out.printf("%d * %d = %d\n", num, i, num * i);
                    delay(1000);
                }
            }
        }

        PrintTable th1 = new PrintTable(2);
        th1.start();
        PrintTable th2 = new PrintTable(4);
        th2.start();

        try {
            th1.join(); // calling thread i.e. main will wait for completion given thread i.e. th1
            th2.join(); // calling thread i.e. main will wait for completion given thread i.e. th2
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

When we start a thread, the thread runs.

When start() is called on a thread, it registers the thread to the scheduler.

When scheduler schedules that thread, then the thread executes run() method.

When run() is completed, the thread is auto terminated.

NOTE: start() does not call run() directly.

Writable | Smart Insert | 32 : 1 : 651

Search | 12:10 PM

Thread life cycle

```
Thread th = new Thread(...);
th.start();

th.join();

th.start(); // exception
```

day19 - demo08/src/com/sunbeam/Program08.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program08.java

```
        }
        @Override
        public void run() {
            try {
                for (int i = 1; i <= 10; i++) {
                    System.out.printf("%d * %d = %d\n", num, i, num * i);
                    Thread.sleep(2000);
                }
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted...");
            }
        } terminate -- run() ends
    }

    PrintTable th = new PrintTable(4);
    th.setPriority(10); // max priority = 10
    System.out.println("th state -- before start() -- " + th.getState()); // NEW
    th.start();

    Scanner sc = new Scanner(System.in);
    System.out.println("Press enter: ");
    sc.nextLine();

    th.interrupt(); // if thread is in sleep(), it will be forcibly wake up i.e. interrupt

    State state = th.getState();
```

Writable | Smart Insert | 30 : 9 : 768

Search 12:42 PM

day19 - demo09/src/com/sunbeam/Program09.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Program09.java × Account.java

```java
import java.util.Iterator;

public class Program09 {
    public static void main(String[] args) throws Exception {
        Account acc = new Account(1, "Saving", 10000);

        class DepositThread extends Thread {
            @Override
            public void run() {
                for (int i = 1; i <= 100; i++) {
                    acc.deposit(100);
                    System.out.println("Balance After Deposit: " + acc.getBalance(
                }
            }
        }
        DepositThread dt = new DepositThread();

        class WithdrawThread extends Thread {
            @Override
            public void run() {
                for (int i = 1; i <= 100; i++) {
                    acc.withdraw(100);
                    System.out.println("Balance After Withdraw: " + acc.getBalance
                }
            }
        }
```

Problems Javadoc Declaration Console ×

<terminated> Program09 [Java Application] C:\Nilesh\setup\sts-4.15.1.RE

```
Balance After Deposit: 11000.0
Balance After Withdraw: 10800.0
Balance After Deposit: 10900.0
Balance After Withdraw: 10800.0
Balance After Deposit: 10900.0
Balance After Withdraw: 10800.0
Balance After Deposit: 10900.0
Balance After Withdraw: 10800.0
Balance After Deposit: 10900.0
Balance After Withdraw: 10800.0
Balance After Deposit: 10900.0
Balance After Withdraw: 10800.0
Balance After Withdraw: 10700.0
Balance After Withdraw: 10600.0
Balance After Withdraw: 10500.0
Balance After Withdraw: 10400.0
Balance After Withdraw: 10300.0
Balance After Withdraw: 10200.0
Balance After Withdraw: 10100.0
Balance After Withdraw: 10000.0
Balance After Withdraw: 9900.0
Final Balance: 9900.0
```

???

Search 12:52 PM

Core Java

Java NIO

- Java NIO (New IO) is an alternative IO API for Java.
- Java NIO offers a different IO programming model than the traditional IO APIs.
- Since Java 7.
- Java NIO enables you to do non-blocking (not fully) IO.
- Java NIO consist of the following core components:
  - Channels e.g. FileChannel, ...
  - Buffers e.g. ByteBuffer, ...
  - Selectors
- Java NIO also provides "helper" classes Paths & Files.
  - exists()
  - ...

Paths and Files

- A Java Path instance represents a path in the file system. A path can point to either a file or a directory. A path can be absolute or relative.

```
Path path = Paths.get("c:\\data\\myfile.txt");
```

- Files class (Files) provides several static methods for manipulating files in the file system.

```
static InputStream newInputStream(Path, OpenOption...) throws IOException;
static OutputStream newOutputStream(Path, OpenOption...) throws IOException;
static DirectoryStream<Path> newDirectoryStream(Path) throws IOException;
static Path createFile(Path, attribute.FileAttribute<?>...) throws
IOException;
static Path createDirectory(Path, attribute.FileAttribute<?>...) throws
IOException;
static void delete(Path) throws IOException;
static boolean deleteIfExists(Path) throws IOException;
static Path copy(Path, Path, CopyOption...) throws IOException;
static Path move(Path, Path, CopyOption...) throws IOException;
static boolean isSameFile(Path, Path) throws IOException;
static boolean isHidden(Path) throws IOException;
static boolean isDirectory(Path, LinkOption...);
static boolean isRegularFile(Path, LinkOption...);
static long size(Path) throws IOException;
static boolean exists(Path, LinkOption...);
static boolean isReadable(Path);
static boolean isWritable(Path);
static boolean isExecutable(Path);
```

```
static List<String> readAllLines(Path) throws IOException;
static Stream<String> lines(Path) throws IOException;
```

Channels and Buffers

- All IO in NIO starts with a Channel. A Channel is similar to IO stream. From the Channel data can be read into a Buffer. Data can also be written from a Buffer into a Channel.

NIO Channels

- Java NIO Channels are similar to IO streams with a few differences:
  - You can both read and write to a Channels. Streams are typically one-way (read or write).
  - Channels can be read and written asynchronously (non-blocking).
  - Channels always read to, or write from, a Buffer.
- Channel Examples
  - FileChannel
  - DatagramChannel // UDP protocol
  - SocketChannel, ServerSocketChannel // TCP protocol

NIO Buffers

- A buffer is essentially a block of memory into which you can write data, which you can then later read again. This memory block is wrapped in a NIO Buffer object, which provides a set of methods that makes it easier to work with the memory block.
- Using a Buffer to read and write data typically follows this 4-step process:
  - Write data into the Buffer
  - Call buffer.flip()
  - Read data out of the Buffer
  - Call buffer.clear() or buffer.compact()
- Buffer Examples
  - ByteBuffer
  - CharBuffer
  - DoubleBuffer
  - FloatBuffer
  - IntBuffer
  - LongBuffer
  - ShortBuffer

Channel and Buffer Example

```
RandomAccessFile aFile = new RandomAccessFile("somefile.txt", "rw");
FileChannel inChannel = aFile.getChannel();

ByteBuffer buf = ByteBuffer.allocate(32);

int bytesRead = inChannel.read(buf); // write data into buffer (from channel)
while (bytesRead != -1) {
```

```
        System.out.println("Read " + bytesRead);
        buf.flip(); // switch buffer from write mode to read mode

        while(buf.hasRemaining()){
            System.out.print((char) buf.get()); // read data from the buffer
        }

        buf.clear(); // clear the buffer
        bytesRead = inChannel.read(buf);
    }
    aFile.close();
```

RandomAccessFile

- RandomAccessFile class from java.io package.
- Capable of reading and writing into a file (on a storage device).
- Internally maintains file read/write position/cursor.
- Homework: Read docs.

Java NIO vs Java IO

- IO: Stream-oriented
- NIO: Buffer-oriented
- IO: Blocking IO
- NIO: Non-blocking IO

Platform Independence

- Java is architecture neutral i.e. can work on various CPU architectures like x86, ARM, SPARC, PPC, etc (if JVM is available on those architectures).
- Java is NOT fully platform independent. It can work on various platforms like Windows, Linux, Mac, UNIX, etc (if JVM is available on those platforms).
- Few features of Java remains platform dependent.
    - Multi-threading (Scheduling, Priority)
    - File IO (Performance, File types, Paths)
    - AWT GUI (Look & Feel)
    - Networking (Socket connection)

Multi-Threading

Inter-thread communication

- wait()
    - Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
    - The current thread must own this object's monitor i.e. wait() must be called within synchronized block/method.
    - The thread releases ownership of this monitor and waits until another thread notifies.
    - The thread then waits until it can re-obtain ownership of the monitor and resumes execution.

- notify()
  - Wakes up a single thread that is waiting on this object's monitor.
  - If multiple threads are waiting on this object, one of them is chosen to be awakened arbitrarily.
  - The awakened thread will not be able to proceed until the current thread relinquishes the lock on this object.
  - This method should only be called by a thread that is the owner of this object's monitor.
- notifyAll()
  - Wakes up all threads that are waiting on this object's monitor.
  - The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object.
  - This method should only be called by a thread that is the owner of this object's monitor.

Member/Nested classes

- By default all Java classes are top-level.
- In Java, classes can be written inside another class/method. They are Member classes.
- Four types of member/nested classes
  - Static member classes -- demo11_01
  - Non-static member class -- demo11_02
  - Local class -- demo11_03
  - Annoymous Inner class -- demo11_04
- When .java file is compiled, separate .class file created for outer class as well as inner class.

Static member classes

- Like other static members of the class (belong to the class, not the object).
- Accessed using outer class (Doesn't need the object of outer class).
- Can access static (private/public) members of the outer class directly.
- Static member class cannot access non-static members of outer class directly.
- The outer class can access all members (including private) of inner class directly (no need of getter/setter).
- The static member classes can be private, public, protected, or default.

```
class Outer {
    private int nonStaticField = 10;
    private static int staticField = 20;

    public static class Inner {
        public void display() {
            System.out.println("Outer.nonStaticField = " + nonStaticField);
// error
            System.out.println("Outer.staticField = " + staticField); // ok
- 20
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Outer.Inner obj = new Outer.Inner();
```

```
            obj.display();
        }
    }
```

Non-static member classes/Inner classes

- Like other non-static members of the class (belong to the object/instance of Outer class).
- Accessed using outer class object (Object of outer class is MUST).
- Can access static & non-static (private) members of the outer class directly.
- The outer class can access all members (including private) of inner class directly (no need of getter/setter).
- The non-static member classes can be private, public, protected, or default.

```
class Outer {
    private int nonStaticField = 10;
    private static int staticField = 20;
    public class Inner {
        public void display() {
            System.out.println("Outer.nonStaticField = " + nonStaticField);
// ok-10
            System.out.println("Outer.staticField = " + staticField); // ok-
20
        }
    }
}
public class Main {
    public static void main(String[] args) {
        //Outer.Inner obj = new Outer.Inner(); // compiler error
        // create object of inner class
            //Outer outObj = new Outer();
            //Outer.Inner obj = outObj.new Inner();
        Outer.Inner obj = new Outer().new Inner();
        obj.display();
    }
}
```

- If Inner class member has same name as of outer class member, it shadows (hides) the outer class member. Such Outer class members can be accessed explicitly using `Outer.this`.

Static member class and Non-static member class -- Application

```
// top-level class
class LinkedList {
    // static member class
    static class Node {
```

```
        private int data;
        private Node next;
        // ...
    }
    private Node head;
    // non-static member class
    class Iterator {
        private Node trav;
        // ...
    }
    // ...
    public void display() {
        Node trav = head;
        while(trav != null) {
            System.out.println(trav.data);
            trav = trav.next;
        }
    }
}
```

Local class

- Like local variables of a method.
- The class scope is limited to the enclosing method.
- If enclosed in static method, behaves like static member class. If enclosed in non-static method, behaves like non-static member class.
- Along with Outer class members, it can also access (effectively) final local variables of the enclosing method.
- We can create any number of objects of local classes within the enclosing method.

```
public class Main {
    private int nonStaticField = 10;
    private static int staticField = 20;
    public static void main(String[] args) {
        final int localVar1 = 1;
        int localVar2 = 2;
        int localVar3 = 3;
        localVar3++;
        // local class (in static method) -- behave like static member class
        class Inner {
            public void display() {
                System.out.println("Outer.nonStaticField = " +
nonStaticField); // error
                System.out.println("Outer.staticField = " + staticField); //
ok 20
                System.out.println("Main.localVar1 = " + localVar1); // ok 1
                System.out.println("Main.localVar2 = " + localVar2); // ok 2
                System.out.println("Main.localVar3 = " + localVar3); //
error
            }
```

```
        }
        Inner obj = new Inner();
        obj.display();
        //new Inner().display();
    }
}
```

Annoymous Inner class

- Creates a new class inherited from the given class/interface and its object is created.
- If in static context, behaves like static member class. If in non-static context, behaves like non-static member class.
- Along with Outer class members, it can also access (effectively) final local variables of the enclosing method.

```
// (named) local class
class EmpnoComparator implements Comparator<Employee> {
    public int compare(Employee e1, Employee e2) {
        return e1.getEmpno() - e2.getEmpno();
    }
}
Arrays.sort(arr, new EmpnoComparator());    // anonymous obj of local class
```

```
// Anonymous inner class
Comparator<Employee> cmp = new Comparator<Employee>() {
    public int compare(Employee e1, Employee e2) {
        return e1.getEmpno() - e2.getEmpno();
    }
};
Arrays.sort(arr, cmp);
```

Date and Time API

- Legacy Java classes
  - Date
  - Calendar
- Java 8 Date Time
  - https://www.baeldung.com/java-8-date-time-intro

Inter-thread communication

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

obj [1]

th1

```
sync(obj)
{
  print("SunBeam");
  obj.notify();
}
```

th2

```
sync(obj)
{
  obj.wait();
  print("INFOTECH");
}
```

obj.wait():

1. unlock the obj
2. block current thread on obj.
3. when wakeup, lock obj again and proceed.

obj.notify();

1. wakeup one of the thread blocked on obj.

obj.notifyAll();

1. wake up all threads blocked on obj.

File Edit Selection View Go ···
You are screen sharing
Stop Share


```Java
RandomAccessFile aFile = new RandomAccessFile("somefile.txt", "rw");
FileChannel inChannel = aFile.getChannel();

ByteBuffer buf = ByteBuffer.allocate(32);

int bytesRead = inChannel.read(buf); // write data into buffer (from channel)
while (bytesRead != -1) {
    System.out.println("Read " + bytesRead);
    buf.flip(); // switch buffer from write mode to read mode

    while(buf.hasRemaining()){
        System.out.print((char) buf.get()); // read data from the buffer
    }

    buf.clear(); // clear the buffer
    bytesRead = inChannel.read(buf);
}
aFile.close();
```

RandomAccessFile

* RandomAccessFile class from java.io package.

main* 0 0 0 0 Java: Ready
Ln 96, Col 24 Spaces: 4 UTF-8 CRLF Markdown
Search
10:04 AM

day20 - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- demo01 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.
  - src
    - com.sunbeam
      - Program01.java
- demo02 [CJ-H-02 main]
  - JRE System Library [JavaSE-1.
  - src

Nested/Member classes:

---

1. Static member classes
2. Non-static member classes
3. Local classes
4. Anonymous inner classes

```
class Outer {
  static class Inner ...
}
```

static member class

1. to create its object, outer class object is not required.

   Outer.Inner obj = new Outer.Inner();

2. Inner class (smc) can access static members of outer class directly or Outer.memberName.

static members:

- class members (not belongs to individual objects)
- to access use ClassName. -- no object needs to be created
- static methods can access static fields directly.

non-static members

- instance members (belongs to individual objects)
- to access use objName. -- object must be created to acce
- non-static methods can access static and non-static field

non-static member class

```
class Outer {
    class Inner { ... }
}
```

1. to create its (nsmc) object, outer class object is mandetory.

   Outer outObj = new Outer();

   Outer.Inner inObj = outObj.new Inner();

2. Inner class (nsmc) can access static as well as non-static members of the outer class directly (using "this") or Outer.this.memberName.

demo02

Search 10:41 AM

1.What is wrapper class? What is their use? Use primitive types as objects used in collection Data-type conversion non-static methods: intValue(), byteValue(), longValue(), floatValue(), doubleValue()
static methods: parseInt(), parseDouble(), valueOf(), ... Helper methods Integer.max(), Integer.sum(), ... Information of primitive types e.g. Integer.BYTES = 4 bytes (int size), Integer.MAX_VALUE (max int), ... Auto-boxing

2.Which are methods of java.lang.Object class? Which are native methods of object class?

3.What is the need of package? Which types are allowed to declare in package? Packages makes Java code modular. It does better organization of the code. Package is a container that is used to group logically related classes, interfaces, enums, and other packages.
Package helps developer: To group functionally related types together. To avoid naming clashing/collision/conflict/ambiguity in source code.
To control the access to types. To make easier to lookup classes in Java source code/docs.

4.Why we can not declare multiple public classes in single .java file? Rule: public class name must be same as file name. Efficient compilation process (Faster linking during import).

5.What is the difference between import and static import? import is used to import public types from other packages. import java.util.ArrayList; import java.io.*;

static import is used to import accessible static members of any class import static java.lang.Math.*; import static java.lang.System.out;

6.What is the difference between checked and unchecked exception? Java compiler expect that certain exception must be handled by the programmer -- checked exception.
catch block throws clause

Most of the checked exceptions arise out of the JVM. Hence Java wants programmer to be aware of them and handle them. For example:
File IO --> IOException Database --> SqlException Threads --> InterruptedException

The other exceptions usually arise due to programmers/users mistakes and within the JVM. For example: NullPointerException ArithmeticException

7.Which are the advantages and disadvantages of generics? Advantages: Type-safety

Disadvantages/Limitations: Doesn't work with primitive types Can't create object/array of generic type Can't overload based on generic type difference Static fields cannot be of generic type param. Can't cast or instanceof with generic type. Can't throw or catch objects of generic type.

8.Type Erasure: The generic type param info is not maintained at runtime (inside JVM). ArrayList list1 = new ArrayList<>(); Internally (inside JVM), it creates ArrayList of "Object"s. And compiler ensures that only Integer can be added to it -- Typesafety. ArrayList list2 = new ArrayList<>();

9.What is difference between Comparable and Comparator?

Comparable The current object (this) is Comparable to the other object -- meant to be written in the class to be compared java.lang.Comparable Natural ordering class Student implements Comparable { // ... public int compareTo(Student other) { int diff = this.roll - other.roll; return diff; } }

Comparator An object compares two other objects. java.util.Comparator Custom order class StudentComparator implements Comparator { public int compare(Student s1, Student s2) { int diff = s1.marks - s2.marks; return diff; } }

10.What is difference between ArrayList and Vector V: Legacy collection 1.0 A: Collection framework 1.2 V: Synchronized -- Thread safe -- Slower -- Suitable in multi-thread applns A: Non-Synchronized -- Non thread safe -- Faster -- Suitable in single-thread applns V: Enumeration (later added support of Iterator) V: Initial size 10 & capacity grow (double) A: Initial size 10 & capacity grow (+half) V: Dynamically growing array A: Dynamically growing array

11.What is serialization and deserialization? What is significance of serialVersionUID? Serialization: Converting state of object into sequence of bytes. It includes state (field values) as well as class info. This state can be stored in file or sent over network. For this, class must be Serializable objectOutputStream.writeObject(obj);

Deserialization: Constructing object back from the state stored as sequence of bytes obj = objectInputStream.readObject();

serialVersionUID class Transaction { static final long serialVersionUID = 2L; // fields ... // methods }

The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization. If the receiver has loaded a class for the object that has a different serialVersionUID than that of the corresponding sender's class, then deserialization will result in an InvalidClassException.

12.Deep copy vs Shallow copy Shallow copy -- Created by Object.clone() method for Cloneable objects Only copies current object (not the embedded objects i.e. objects referenced by fields in the object). class Human implements Cloneable { int age; String name; Date birth; // ... @Override // shallow copy Object clone() throws ... { Human other = super.clone(); // Object.clone() return other; } }

Copies the whole object including embedded objects. Needs to be done explicitly in overridden clone() method class Human implements Cloneable { int age; String name; Date birth; // ... @Override // deep copy Object clone() throws ... { Human other = super.clone(); // Object.clone() other.birth = this.birth.clone(); return other; } }

13.difference between Closeable and finalize(). When GC collects unused object, it will invoke finalize() method. The finalize() should be overridden in your class to cleanup the resources class MyDao { Connection con; public void finalize() { con.close(); } }

However, you cannot force/gurantee the GC. Possibly if GC is delayed, your resource remains open for longer duration. Poor performance. Closeable implementation ensure that resource can be closed immediatly after its use. class MyDao implements Closeable { Connection con; // ... public void close() { con.close(); } }

MyDao dao = new MyDao(); // ... dao.close(); // resource closed immediatly

14.What is functional interface? Which functional interfaces are predefined and where they are used? Functional interface: SAM (Single Abstract Method)
Any number of default methods Any number of static methods

@FunctionalInterface -- compiler check for SAM - if 0 or multiple SAM, then compiler error.

Functional Interface: Comparable, Comparator, Runnable, Closeable, ...

(In Java 8) Functional Interfaces -- java.util.functional.*

Predicate ... boolean test(T val);
used in filter() operation

Function ... R apply(T val); used in map() operation

Consumer ... void accept(T val); used in forEach() operation

Supplier ... T get(); used in generator

Functional Interfaces are also used for method references. Consumer cons = System.out::println;
cons.accept("Sunbeam"); // --> System.out.println("Sunbeam");

15. What is significance of filter(), map(), flatMap() and reduce() operations? In which scenarios they are used?

*java.util.Stream -- represents stream of data elements No storage: Stream doesn't hold data like collections. It processes data from collections/arrays.
Immutable: stream1 --> Intermediate operation --> stream2 Lazily evalutated:
stream1.iop1().iop2().iop3().iop4().iop5().terminal();
Not reusable: Can perform only one terminal operation on stream.

Stream operations Intermediate operations e.g. map(), filter(), sort(), ... Terminal operations e.g. forEach(), collect(), reduce(), ...

```
Integer[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9};
Stream<Integer> stream = Stream.of(array);
//stream ->     // 1, 2, 3, 4, 5, 6, 7, 8, 9
   .filter(i -> i % 2 != 0)
               // 1, 3, 5, 7, 9
  .map(i -> "DAC"+i)
               // DAC1, DAC3, DAC5, DAC7, DAC9
  .forEach(s -> System.out.println(s));
```

https://winterbe.com/posts/2014/03/16/java-8-tutorial/ (https://winterbe.com/posts/2014/03/16/java-8-tutorial/)
https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/ (https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/)

1.What is wrapper class? What is their use? Use primitive types as objects used in collection Data-type conversion non-static methods: intValue(), byteValue(), longValue(), floatValue(), doubleValue()
static methods: parseInt(), parseDouble(), valueOf(), ... Helper methods Integer.max(), Integer.sum(), ... Information of primitive types e.g. Integer.BYTES = 4 bytes (int size), Integer.MAX_VALUE (max int), ... Auto-boxing

2.Which are methods of java.lang.Object class? Which are native methods of object class?

3.What is the need of package? Which types are allowed to declare in package? Packages makes Java code modular. It does better organization of the code. Package is a container that is used to group logically related classes, interfaces, enums, and other packages.
Package helps developer: To group functionally related types together. To avoid naming clashing/collision/conflict/ambiguity in source code.
To control the access to types. To make easier to lookup classes in Java source code/docs.

4.Why we can not declare multiple public classes in single .java file? Rule: public class name must be same as file name. Efficient compilation process (Faster linking during import).

5.What is the difference between import and static import? import is used to import public types from other packages. import java.util.ArrayList; import java.io.*;

static import is used to import accessible static members of any class import static java.lang.Math.*; import static java.lang.System.out;

6.What is the difference between checked and unchecked exception? Java compiler expect that certain exception must be handled by the programmer -- checked exception.
catch block throws clause

Most of the checked exceptions arise out of the JVM. Hence Java wants programmer to be aware of them and handle them. For example:
File IO --> IOException Database --> SqlException Threads --> InterruptedException

The other exceptions usually arise due to programmers/users mistakes and within the JVM. For example: NullPointerException ArithmeticException

7.Which are the advantages and disadvantages of generics? Advantages: Type-safety

Disadvantages/Limitations: Doesn't work with primitive types Can't create object/array of generic type Can't overload based on generic type difference Static fields cannot be of generic type param. Can't cast or instanceof with generic type. Can't throw or catch objects of generic type.

8.Type Erasure: The generic type param info is not maintained at runtime (inside JVM). ArrayList list1 = new ArrayList<>(); Internally (inside JVM), it creates ArrayList of "Object"s. And compiler ensures that only Integer can be added to it -- Typesafety. ArrayList list2 = new ArrayList<>();

9.What is difference between Comparable and Comparator?

Comparable The current object (this) is Comparable to the other object -- meant to be written in the class to be compared java.lang.Comparable Natural ordering class Student implements Comparable { // ... public int compareTo(Student other) { int diff = this.roll - other.roll; return diff; } }

Comparator An object compares two other objects. java.util.Comparator Custom order class StudentComparator implements Comparator { public int compare(Student s1, Student s2) { int diff = s1.marks - s2.marks; return diff; } }

10.What is difference between ArrayList and Vector V: Legacy collection 1.0 A: Collection framework 1.2 V: Synchronized -- Thread safe -- Slower -- Suitable in multi-thread applns A: Non-Synchronized -- Non thread safe -- Faster -- Suitable in single-thread applns V: Enumeration (later added support of Iterator) V: Initial size 10 & capacity grow (double) A: Initial size 10 & capacity grow (+half) V: Dynamically growing array A: Dynamically growing array

11.What is serialization and deserialization? What is significance of serialVersionUID? Serialization: Converting state of object into sequence of bytes. It includes state (field values) as well as class info. This state can be stored in file or sent over network. For this, class must be Serializable objectOutputStream.writeObject(obj);

Deserialization: Constructing object back from the state stored as sequence of bytes obj = objectInputStream.readObject();

serialVersionUID class Transaction { static final long serialVersionUID = 2L; // fields ... // methods }

The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization. If the receiver has loaded a class for the object that has a different serialVersionUID than that of the corresponding sender's class, then deserialization will result in an InvalidClassException.

12.Deep copy vs Shallow copy Shallow copy -- Created by Object.clone() method for Cloneable objects Only copies current object (not the embedded objects i.e. objects referenced by fields in the object). class Human implements Cloneable { int age; String name; Date birth; // ... @Override // shallow copy Object clone() throws ... { Human other = super.clone(); // Object.clone() return other; } }

Copies the whole object including embedded objects. Needs to be done explicitly in overridden clone() method class Human implements Cloneable { int age; String name; Date birth; // ... @Override // deep copy Object clone() throws ... { Human other = super.clone(); // Object.clone() other.birth = this.birth.clone(); return other; } }

13.difference between Closeable and finalize(). When GC collects unused object, it will invoke finalize() method. The finalize() should be overridden in your class to cleanup the resources class MyDao { Connection con; public void finalize() { con.close(); } }

However, you cannot force/gurantee the GC. Possibly if GC is delayed, your resource remains open for longer duration. Poor performance. Closeable implementation ensure that resource can be closed immediatly after its use. class MyDao implements Closeable { Connection con; // ... public void close() { con.close(); } }

MyDao dao = new MyDao(); // ... dao.close(); // resource closed immediatly

14.What is functional interface? Which functional interfaces are predefined and where they are used? Functional interface: SAM (Single Abstract Method)
Any number of default methods Any number of static methods

@FunctionalInterface -- compiler check for SAM - if 0 or multiple SAM, then compiler error.

Functional Interface: Comparable, Comparator, Runnable, Closeable, ...

(In Java 8) Functional Interfaces -- java.util.functional.*

Predicate ... boolean test(T val);
used in filter() operation

Function ... R apply(T val); used in map() operation

Consumer ... void accept(T val); used in forEach() operation

Supplier ... T get(); used in generator

Functional Interfaces are also used for method references. Consumer cons = System.out::println;
cons.accept("Sunbeam"); // --> System.out.println("Sunbeam");

15. What is significance of filter(), map(), flatMap() and reduce() operations? In which scenarios they are used?

*java.util.Stream -- represents stream of data elements No storage: Stream doesn't hold data like collections. It processes data from collections/arrays.
Immutable: stream1 --> Intermediate operation --> stream2 Lazily evalutated:
stream1.iop1().iop2().iop3().iop4().iop5().terminal();
Not reusable: Can perform only one terminal operation on stream.

Stream operations Intermediate operations e.g. map(), filter(), sort(), ... Terminal operations e.g. forEach(), collect(), reduce(), ...

```
Integer[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9};
Stream<Integer> stream = Stream.of(array);
//stream ->     // 1, 2, 3, 4, 5, 6, 7, 8, 9
   .filter(i -> i % 2 != 0)
               // 1, 3, 5, 7, 9
  .map(i -> "DAC"+i)
               // DAC1, DAC3, DAC5, DAC7, DAC9
  .forEach(s -> System.out.println(s));
```

https://winterbe.com/posts/2014/03/16/java-8-tutorial/ (https://winterbe.com/posts/2014/03/16/java-8-tutorial/)
https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/ (https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/)