Core Java

Day 02 Agenda

- Java Buzzwords
- PATH vs CLASSPATH
- Input/Output
  - Console
  - Scanner
- Language Fundamentals
  - Naming conventions
  - Comments
  - Keywords
  - Data types
  - Literals
  - Variables
  - Operators
- Wrapper classes
  - Auto-Boxing

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.

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

```java
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:

    ```
    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

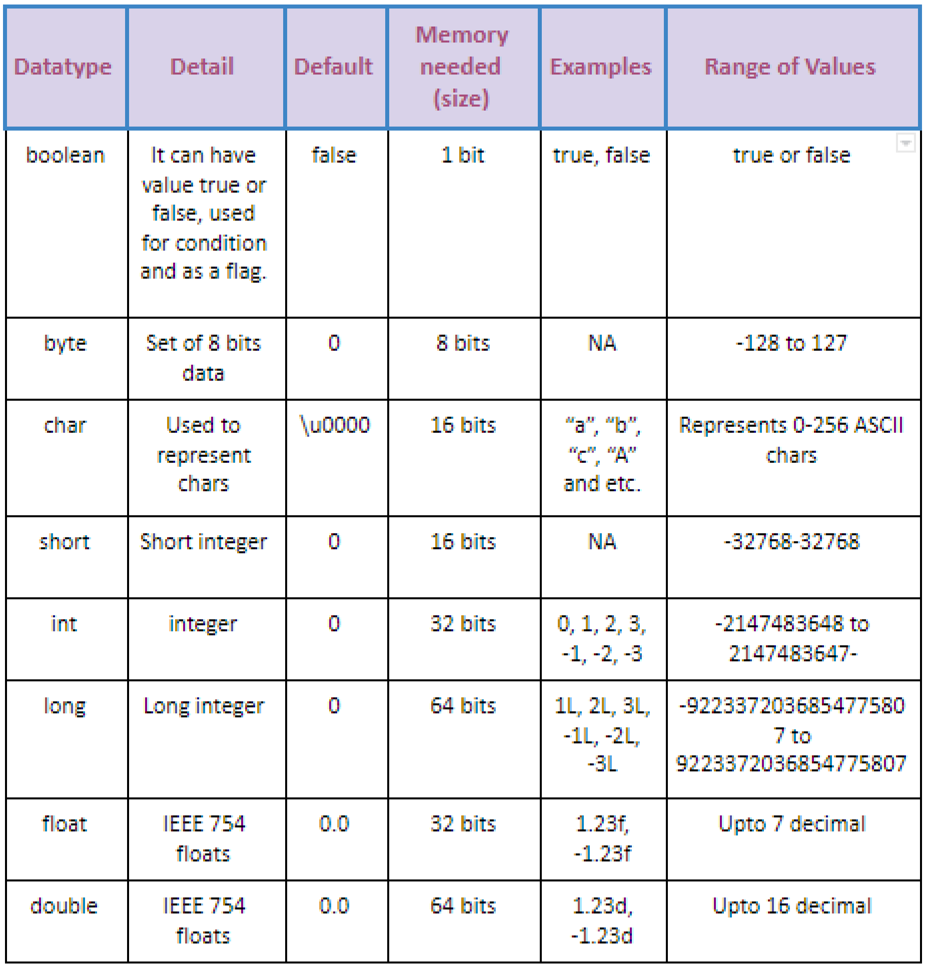
```
Data types
    |- Primitive types (Value types)
    |      |- Boolean: boolean
    |      |- Character: char
    |      |- Integral: byte, short, int, long
    |      |- Floating-point: float, double
    |
    |- Non-Primitive types (Reference types)
           |- class
           |- interface
           |- 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	-922337203685477580 7 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 langaugues.
- 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;
```

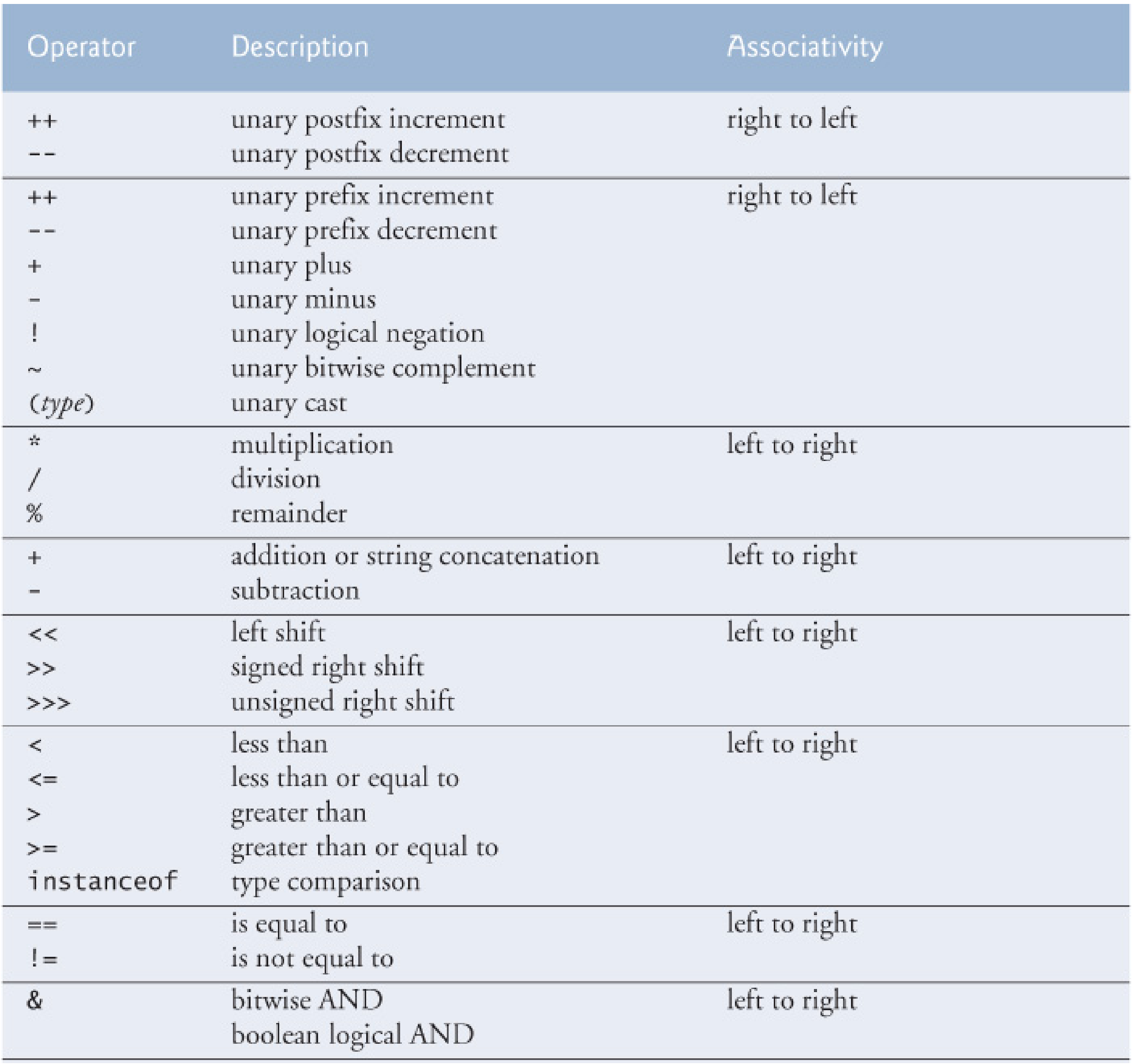
Variables

- 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

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

  - Convert types

    ```
    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

    ```
    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

    ```
    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
  ```

Assignments

1. Write a program to calculate area and circumference of Circle. Radius should be initialized with fixed value. Use any editor to type the code and compile on command line.

2. Write a program to calculate area and perimeter of Square. Side should be initialized with fixed value. Use Eclipse STS 4.x IDE.

3. Write a program to perform following operations on double values.

   - Maximum from two double values (input from user).
   - Minimum from two double values (input from user).
   - Sum of two double values (input from user).
   - Input a string and convert to double.
   - Input a double from user and convert to int.

  - Print size of "double" type in bytes.
  - Hint: use wrapper class "Double". https://docs.oracle.com/javase/8/docs/api/java/lang/Double.html