Java Workshop

DDUC ACM STUDENT CHAPTER
DEEN DAYAL UPADHYAYAYA COLLEGE
UNIVERSITY OF DELHI

Exploring java.lang

FUNDAMENTAL TO VIRTUALLY ALL OF JAVA PROGRAMMING

java.lang

- Java's most widely used package.
- Automatically imported into all programs.
- Contains classes and interfaces that are fundamental to Java programming.
- Still providing a lot of depreciated methods in several classes to support an ever shrinking pool of legacy code.
- Also holds a child package *reflect* which is also very important and powerful. This package holds **Array**, **Constructor**, **Method**, and **Field** classes.

Important Classes and Packages in java.lang

Classes				
Boolean	Float	Package	StrictMath	ThreadGroup
Byte	Integer	Process	String	Throwable
Character	Long	ProcessBuilder	StringBuffer	Void
Class	Math	Runtime	StringBuilder	
Double	Number	SecurityManager	System	
Enum	Object	Short	Thread	

Interfaces		
Appendable	Cloneable	Readable
AutoClosable	Comparable	Runnable
CharSequence	Iterable	

Primitive Type Wrappers

- Java uses primitive types, such as **int** or **char** for performance reasons.
- These data type are not part of object hierarchy.
- They are passed by value and cannot be directly passed by reference.
- There is no way for two methods to refer to the ***same instance*** of a primitive type.
- To store primitive types as objects, wrappers are needed and that's why Java provides wrapper classes for all primitive types.
- These classes encapsulate or wrap primitive types within a class and are commonly referred as ***type wrappers***.

The Number Class

- Abstract class **Number** defines a superclass which is implemented by the classes that wrap numeric types **byte**, **short**, **int**, **long**, **float**, and **double**.
- **Number** has abstract methods that return the value of the object in each of the different number formats. For example **doubleValue()** returns the value as a **double**.
- These methods are:
  - byte byteValue()
  - double doubleValue()
  - float floatValue()
  - int intValue()
  - long longValue()
  - short shortValue()
    - The value returned by these methods might be rounded or truncated.
- **Number** has concrete subclasses that hold explicit values of each numeric type: **Double**, **Float**, **Byte**, **Short**, **Integer**, and **Long**.

The Double and Float Classes

- These classes are wrappers for floating point values of **double** and **float**.
- The constructors of **Float** class are:
  - Float(double *num*)
  - Float(float *num*)
  - Float(String *str*) throws NumberFormatException
- The constructors of **Double** class are:
  - Double(double *num*)
  - Double(String *str*) throws NumberFormatException

Constants defined in Float Class

Constant	Description	Value
MAX_EXPONENT	Maximum exponent	127
MAX_VALUE	Maximum positive value	3.4028235e38
MIN_EXPONENT	Minimum exponent	-126
MIN_NORMAL	Minimum positive normal value	1.17549435e-38
MIN_VALUE	Minimum positive value	1.4e-45
NaN	Not a number	NaN
POSITIVE_INFINITY	Positive infinity	Infinity
NEGATIVE_INFINITY	Negative infinity	-Infinity
SIZE	The bit width of the wrapped value	32
TYPE	The **Class** object for **float** or **double**	float

Important Methods Defined by Float Class

- byte byteValue()
- static int compare(float *num1*, float *num2*)
  - Returns 0 if *num1* and *num2* are equal, -1 if *num1* is smaller and 1 if *num1* is larger.
- int compareTo(Float *f*)
- double doubleValue()
- boolean equals(Object *floatObj*)
- float floatValue()
- int intValue()
- boolean isInfinite()
- static boolean isInfinite(float *num*)

Important Methods Defined by Float Class

- boolean isNaN()
- static boolean isNaN (float *num*)
- long longValue()
- static float parseFloat(String *str*)
  - throws NumberFormatException
- short shortValue()
- static String toHexString(float *num*)
- String toString()
- static String toString(float *num*)
- static Float valueOf(float *num*)
- static Float valueOf(String *str*)
  - throws NumberFormatException

Constants defined in Double Class

Constant	Description	Value
MAX_EXPONENT	Maximum exponent	1023
MAX_VALUE	Maximum positive value	1.7976931348623157e308
MIN_EXPONENT	Minimum exponent	-1022
MIN_NORMAL	Minimum positive normal value	2.2250738585072014e-308
MIN_VALUE	Minimum positive value	4.9e-324
NaN	Not a number	NaN
POSITIVE_INFINITY	Positive infinity	Infinity
NEGATIVE_INFINITY	Negative infinity	-Infinity
SIZE	The bit width of the wrapped value	64
TYPE	The **Class** object for **float** or **double**	double

Important Methods Defined by Double Class

- byte byteValue()
- static int compare(double *num1*, double *num2*)
  - Returns 0 if *num1* and *num2* are equal, -1 if *num1* is smaller and 1 if *num1* is larger.
- int compareTo(Double *d*)
- double doubleValue()
- boolean equals(Object *doubleObj*)
- float floatValue()
- int intValue()
- boolean isInfinite()
- static boolean isInfinite(float *num*)

Important Methods Defined by Double Class

- boolean isNaN()
- static boolean isNaN (float *num*)
- long longValue()
- static float parseDouble(String *str*)
  - throws NumberFormatException
- short shortValue()
- static String toHexString(double *num*)
- String toString()
- static String toString(double *num*)
- static Double valueOf(double *num*)
- static Double valueOf(String *str*)
  - throws NumberFormatException

Byte, Short, Integer, and Long Classes

- These classes are wrappers for floating point values of **byte**, **short**, **int**, and **long** respectively.
- The constructors of **Byte** class are:
  - Byte(byte *num*)
  - Byte(String *str*) throws NumberFormatException
- The constructors of **Short** class are:
  - Short(short *num*)
  - Short(String *str*) throws NumberFormatException
- The constructors of **Integer** class are:
  - Integer(int *num*)
  - Integer(String *str*) throws NumberFormatException
- The constructors of **Long** class are:
  - Long(long *num*)
  - Long(String *str*) throws NumberFormatException

Constants defined in Byte Class

Constant	Description	Value
MAX_VALUE	Maximum positive value	127
MIN_VALUE	Minimum positive value	-128
SIZE	The bit width of the wrapped value	8
TYPE	The **Class** object for **byte**, **short**, **int** or **long**	byte

Important Methods Defined by Byte Class

- byte byteValue()
- static int compare(byte *num1*, byte *num2*)
  - Returns 0 if *num1* and *num2* are equal, -1 if *num1* is smaller and 1 if *num1* is larger.
- int compareTo(Byte *b*)
- static Byte decode(String *str*)
  - throws NumberFormatException
- double doubleValue()
- boolean equals(Object *byteObj*)
- float floatValue()
- int intValue()
- long longValue()

Important Methods Defined by Byte Class

- static byte parseByte(String *str*)
  - throws NumberFormatException
- static byte parseByte(String *str*, int *radix*)
  - throws NumberFormatException
- short shortValue()
- String toString()
- static String toString(byte *num*)
- static Byte valueOf(byte *num*)
- static Byte valueOf(String *str*)
  - throws NumberFormatException
- static Byte valueOf(String *str*, int *radix*)
  - throws NumberFormatException

Constants defined in Short Class

Constant	Description	Value
MAX_VALUE	Maximum positive value	32767
MIN_VALUE	Minimum positive value	-32768
SIZE	The bit width of the wrapped value	16
TYPE	The **Class** object for **byte**, **short**, **int** or **long**	short

Important Methods Defined by Short Class

- byte byteValue()
- static int compare(short *num1*, short *num2*)
  - Returns 0 if *num1* and *num2* are equal, -1 if *num1* is smaller and 1 if *num1* is larger.
- int compareTo(Short *s*)
- static Short decode(String *str*)
  - throws NumberFormatException
- double doubleValue()
- boolean equals(Object *shortObj*)
- float floatValue()
- int intValue()
- long longValue()

Important Methods Defined by Short Class

- static short parseShort(String *str*)
  - throws NumberFormatException
- static short parseShort(String *str*, int *radix*)
  - throws NumberFormatException
- static short reverseBytes(short *num*)
- short shortValue()
- String toString()
- static String toString(short *num*)
- static Short valueOf(short *num*)
- static Short valueOf(String *str*)
  - throws NumberFormatException
- static Short valueOf(String *str*, int *radix*)
  - throws NumberFormatException

Constants defined in Integer Class

Constant	Description	Value
MAX_VALUE	Maximum positive value	2147483647
MIN_VALUE	Minimum positive value	-2147483648
SIZE	The bit width of the wrapped value	32
TYPE	The **Class** object for **byte**, **short**, **int** or **long**	int

Important Methods Defined by Integer Class

- static int bitCount(int *num*)
- byte byteValue()
- static int compare(int *num1*, int *num2*)
  - Returns 0 if *num1* and *num2* are equal, -1 if *num1* is smaller and 1 if *num1* is larger.
- int compareTo(Integer *i*)
- static Integer decode(String *str*)
  - throws NumberFormatException
- double doubleValue()
- boolean equals(Object *integerObj*)
- float floatValue()
- static int highestOneBit(int *num*)
- static int lowestOneBit(int *num*)

Important Methods Defined by Integer Class

- int intValue()
- long longValue()
- static int numberOfLeadingZeros(int *num*)
- static int numberOfTrailingZeros(int *num*)
- static int parseInt (String *str*)
  - throws NumberFormatException
- static int parseInt(String *str*, int *radix*)
  - throws NumberFormatException
- static int reverse(int *num*)
- static int reverseBytes(int *num*)
- static int rotateLeft(int *num*, int *n*)
- static int rotateRight(int *num*, int *n*)

Important Methods Defined by Integer Class

- static int signum(int *num*)
- short shortValue()
- static String toBinaryString(int *num*)
- static String toHexString(int *num*)
- static String toOctalString(int *num*)
- String toString()
- static String toString(int *num*)
- static Integer valueOf(int *num*)
- static Integer valueOf(String *str*)
  - throws NumberFormatException
- static Integer valueOf(String *str*, int *radix*)
  - throws NumberFormatException

Constants defined in Long Class

Constant	Description	Value
MAX_VALUE	Maximum positive value	9223372036854775807
MIN_VALUE	Minimum positive value	-9223372036854775808
SIZE	The bit width of the wrapped value	64
TYPE	The **Class** object for **byte**, **short**, **int** or **long**	long

Important Methods Defined by Long Class

- static int bitCount(long *num*)
- byte byteValue()
- static int compare(long *num1*, long *num2*)
  - Returns 0 if *num1* and *num2* are equal, -1 if *num1* is smaller and 1 if *num1* is larger.
- int compareTo(Long *l*)
- static Long decode(String *str*)
  - throws NumberFormatException
- double doubleValue()
- boolean equals(Object *longObj*)
- float floatValue()
- static int highestOneBit(long *num*)
- static int lowestOneBit(long *num*)

Important Methods Defined by Long Class

- int intValue()
- long longValue()
- static int numberOfLeadingZeros(long *num*)
- static int numberOfTrailingZeros(long *num*)
- static long parseLong (String *str*)
  - throws NumberFormatException
- static long parseLong(String *str*, int *radix*)
  - throws NumberFormatException
- static long reverse(long *num*)
- static long reverseBytes(long *num*)
- static long rotateLeft(long *num*, int *n*)
- static long rotateRight(long *num*, int *n*)

Important Methods Defined by Long Class

- static int signum(long *num*)
- short shortValue()
- static String toBinaryString(long *num*)
- static String toHexString(long *num*)
- static String toOctalString(long *num*)
- String toString()
- static String toString(long *num*)
- static Long valueOf(long *num*)
- static Long valueOf(String *str*)
  - throws NumberFormatException
- static Long valueOf(String *str*, int *radix*)
  - throws NumberFormatException

Character Class

- **Character** class is a simple wrapper around a **char**.
- The constructor for **Character** class is:
  - Character(char *ch*)
- Several constants defined in **Character** class are:

Constant	Description	Value
MAX_RADIX	The largest radix	Char with ASCII Code 65535
MIN_RADIX	The smallest radix	Char with ASCII Code 0
MAX_VALUE	The largest character value	36
MIN_VALUE	The largest character value	2
SIZE	The bit width of wrapped value	
TYPE	The **Class** object for **char**	char

Methods Defined in Character Class

- char charValue()
- static char forDigit(int *num*, int *radix*)
- static int digit(char *digit*, int *radix*)
- static int compare(char *char1*, char *char2*)
  - Returns 0 if *char1* and *char2* are equal, -1 if *char1* is smaller and 1 if *char1* is larger.
- int compareTo(Character *c*)
- boolean equals(Object characterObject)
- static boolean isDigit(char *ch*)
- static boolean isLetter(char *ch*)
- static boolean isLetterOrDigit(char *ch*)
- static boolean isLowerCase(char *ch*)
- static boolean isUpperCase(char *ch*)
- static boolean isWhiteSpace(char *ch*)
- static char toLowerCase(char *ch*)
- static char toUpperCase(char *ch*)

Boolean Class

- **Boolean** class is a very thin wrapper around **boolean** values, which are generally used when passing a **boolean** variable by reference.
- The constructors for **Boolean** class are:
  - Boolean(boolean *boolValue*)
  - Boolean(String *boolString*)
- **Boolean** class defines constants **TRUE** and **FALSE**, which define true and false **Boolean** objects.
- **Boolean** class has another constant **TYPE** also which is the **Class** object for **boolean**.

Methods Defined in Boolean Class

- boolean booleanValue()
- static int compare(boolean *b1*, boolean *b2*)
  - Returns 0 if *b1* and *b2* are equal, -1 if *b1* is false and *b2* is true, and 1 if *b1* is true and *b2* is false.
- int compareTo(boolean *b*)
- boolean equals(Object *booleanObject*)
- static boolean parseBoolean(String *str*)
- String toString()
- static String toString(boolean *booleanValue*)
- static Boolean valueOf(boolean *booleanValue*)
- static Boolean valueOf (String *str*)

Void Class

- **Void** class has one field, **TYPE** and one constructor Void() only.
- **TYPE** holds a reference to the **Class** object for type **void**.

Process Class

- **Process** is an abstract class which encapsulates a *process*.
- It is used mainly as a superclass for the type of objects created by **exec()** in **Runtime** class.
- It is also used as a superclass for the type of objects created by **start()** in the **ProcessBuilder** class.

Methods defined in Process Class

- **Process** class provides following abstract methods:
  - void destroy()
    - Terminates the process.
  - int exitValue()
    - Returns an exit code obtained from a subprocess.
  - InputStream getErrorStream()
    - Returns an input stream that reads input from the process's **err** output stream.
  - InputStream getInputStream()
    - Returns an input stream that reads input from the process's **out** output stream.
  - OutputStream getOutputStream()
    - Returns an input stream that writes output to the process's **in** input stream.
  - int waitFor() throws InterruptedException
    - Returns the exit code returned by the process. This method returns only after termination of the process.

Runtime Class

- **Runtime** class encapsulates the run-time environment.
- A **Runtime** object cannot be instantiated but a reference to the current **Runtime** object can be taken by static method **Runtime.getRuntime()**.
- By using this reference, several methods can be called to control the state and behavior of JVM.
- Applets cannot call any of the **Runtime** methods without raising a **SecurityException**.

Methods Defined in Runtime Class

- Some **Runtime** methods are:
  - static Runtime getRuntime()
    - Returns the runtime object associated with the current Java application.
  - int availableProcessors()
    - Returns the number of processors available to the Java virtual machine.
  - Process exec(String *progName*) throws IOException
    - Executes a program specified by *progName* as a separate process and an object describing that process is returned.
  - Process exec(String *progName*, String *env[]*) throws IOException
    - Exceutes a program specified by *progName* and environment specified by *env*
  - Process exec(String *cmdArray[]*) throws IOException
    - Executes the command line specified by strings in *cmdArray*.
  - Process exec(String *cmdArray[]*, String *env[]*) throws IOException
    - Exceutes the coomand line specified by *progName* and environment specified by *env*.

Methods Defined in Runtime Class

- void exit(int *exitCode*)
  - Halts executiton and returns the value of *exitCode* to the parent process.
- long freeMemory()
  - Returns the approximate number of bytes of free memory available to Java run-time system.
- void gc()
  - Initiates the garbage collection.
- long totalMemory()
  - Returns the total number of bytes of memory available to the program.

Memory Management

- Java provides automatic garbage collection, but sometimes it is need to know how large the object heap is and how much of it is left.
- This information can be used to check the code for efficiency or to know how much objects can be instantiated in future.
- For this purpose **freeMemory()**, **totalMemory()**, and **gc()** methods are used.

System Class

- **System** class holds a collection of static methods and variables.
- The standard input, output, and error output of Java run time are stored in the **in**, **out**, and **err** variables.
- Many methods of **System** class throw **SecurityException** if the operation is not permitted by security manager
- Only constructor of this class is declared private so instantiation of this class is not possible.

Methods Defined in System Class

- Some methods of **System** class are:
  - static void arraycopy(Object *src*, int *srcStart*, Object *dest*, int *destStart*, int *size*)
    - Copies *size* elements of array *src* starting from *srcStart* into array *dest* starting from *destStart*.
    - Throws IndexOutOfBoundsException if copying would cause access of data outside array bounds.
    - Throws ArrayStoreException if an element in the *src* array could not be stored into the *dest* array because of a type mismatch.
    - Throws NullPointerException if either *src* or *dest* is null.
  - static String clearProperty(String *which*)
    - Deletes the environmental variable specified by *which* and returns previous value associated with *which*. It may throw SecurityException, NullPointerException, or IllegalArgumentException.

Methods Defined in System Class

- static long currentTimeMillis()
  - Returns the current time in terms of milliseconds since midnight, January 1, 1970.
  - While the unit of time of the return value is a millisecond, the granularity of the value depends on the underlying operating system and may be larger.
- static void exit(int *exitCode*)
  - Halts executiton and returns the value of *exitCode* to the parent process (usually operating system). 0 means normal termination.
- static void gc()
  - Initiates garbage collection.

Methods Defined in System Class

- static String getenv(String *which*)
  - Returns the value associated with the environmental variable passed in *which*.
- static SecurityManager getSecurityManager()
  - Returns the current security manager or a null object if no security manager is installed.
- static void setSecurityManager(SecurityManager *secMan*)
  - Sets security manager to the specified by *secMan*.
- static String lineSeparator()
  - Returns a string that contains the line-separator characters.
- static long nanoTime()
  - Obtains the most precise timer in system and returns its value in terms of nanoseconds since some arbitrary starting point. The accuracy of the timer is unknowable.

Using currentTimeMillis() or nanoTime()

```
// Timing program execution.
class Elapsed {
    public static void main(String args[]) {
        long start, end;
        System.out.println("Timing a for loop from 0 to 1,000,000");
        // time a for loop from 0 to 1,000,000
        start = System.currentTimeMillis(); // get starting time
        //or use start = System.nanoTime();
        for(int i=0; i < 1000000; i++) ;
        end = System.currentTimeMillis(); // get ending time
        //or use end = System.nanoTime();
        System.out.println("Elapsed time: " + (end-start));
    }
}
```

Using arraycopy()

```
// Using arraycopy().
class ACDemo {
    static byte a[] = { 65, 66, 67, 68, 69, 70, 71, 72, 73, 74 };
    static byte b[] = { 77, 77, 77, 77, 77, 77, 77, 77, 77, 77 };

    public static void main(String args[]) {
        System.out.println("a = " + new String(a));
        System.out.println("b = " + new String(b));
        System.arraycopy(a, 0, b, 0, a.length);
        System.out.println("a = " + new String(a));
        System.out.println("b = " + new String(b));
        System.arraycopy(a, 0, a, 1, a.length - 1);
        System.arraycopy(b, 1, b, 0, b.length - 1);
        System.out.println("a = " + new String(a));
        System.out.println("b = " + new String(b));
    }
}
```

Using arraycopy()

- Output of previous example would be:

  a = ABCDEFGHIJ

  b = MMMMMMMMMM

  a = ABCDEFGHIJ

  b = ABCDEFGHIJ

  a = AABCDEFGHI

  b = BCDEFGHIJJ

Object Class

- **Object** is a superclass of all other classes.
- Some of the methods defined in **Object** class are:
  - Object clone() throws CloneNotSupportedEception
    - Creates a new object that is same as the invoking object.
  - boolean equal(Object *object*)
    - Returns **true** if the invoking object is equivalent to *object*.
  - final Class<?> getClass()
    - Obtains a **Class** object that describes the invoking object.
  - String toString()
    - Returns a string that describes the object.

The **Class** Class

- **Class** encapsulates the run-time state of a class or interface.
- Objects of type **Class** are created automatically, when classes are loaded.
- A **Class** object cannot be explicitly created.
- A **Class** object can be obtained by calling the **getClass()** method defined by **Object**.
- **Class** is a generic type that is declared as:
  - class Class<T>
  - Here, **T** is the type of the class or interface represented.

Methods Defined in **Class** class

- static Class<?> forName(String *name*)
  - Returns a **Class** object given its complete name. it throws **ClassNotFoundException**.
- Constructor<?>[] getConstructors()
  - Obtains a **Constructor** object for each public constructor of the class represented by invoking object and stores them in an array. It throws **SecurityException**.
- Constructor<?>[] getDeclaredConstructors()
  - Obtains a **Constructor** object for each constructor declared by the class represented by invoking object and stores them in an array. Superclass constructors are ignored. It throws **SecurityException**.

Methods Defined in **Class** Class

- Field[] getFields()
  - Obtains a **Field** object for each public field declared by the class or interface represented by the invoking object and stores them in an array. It throws **SecurityException**.
- Field[] getDeclaredFields()
  - Obtains a **Field** object for each field declared by the class or interface represented by the invoking object and stores them in an array. Inherited fields are ignored. It throws **SecurityException.**
- Method[] getMethods()
  - Obtains a **Method** object for each public method declared by the class or interface represented by the invoking object and stores them in an array. It throws **SecurityException.**
- Method[] getDeclaredMethods()
  - Obtains a **Method** object for each method declared by the class or interface represented by the invoking object and stores them in an array. Inherited methods are ignored. It throws **SecurityException.**

Methods Defined in **Class** Class

- Class<? super T> getSuperclass()
  - Returns the superclass of the type represented by the invoking object. The return value is **null** if the represented type is **Object** or not a class.
- boolean isInterface()
  - Returns **true** if the type represented by the invoking object is an interface. Otherwise, it returns **false**.
- T newInstance()
  - Creates a new instance that is of the same type as that represented by invoking object. This is equivalent to using **new** with the class' default constructor. The new object is returned and this method fails if the represented type is abstract, not a class, or does not have a default constructor.
  - throws **IllegalAccessException** and **InstantiationException**
- String toString()
  - Returns the string representation of the type represented by the invoking object or interface.

Math Class

- The **Math** class contain all the floating-point functions that are used for geometry and trigonometry, as well several general-purpose methods.
- It has private constructor to prevent its instantiation.
- **Math** class defines two **double** constants:
  - **E** (2.718281828459045235 4)
  - **PI** (3.14159265358979323846)

Trigonometric Functions in Math Class

- static double sin(double *arg*)
- static double cos(double *arg*)
- static double tan(double *arg*)
- static double asin(double *arg*)
- static double acos(double *arg*)
- static double atan(double *arg*)
- static double atan2(double *x*, double *y*)
- static double sinh(double *arg*)
- static double cosh(double *arg*)
- static double tanh(double *arg*)

Exponential Functions in Math Class

- static double sqrt(double *arg*) : square root of *arg*
- static double cbrt(double *arg*) : cube root of *arg*
- static double exp(double *arg*) : e to the power *arg*
- static double expm1(double *arg*): e to the power *arg*-1
- static double log(double *arg*) : natural log of *arg*
- static double log10(double *arg*) : base 10 log of *arg*
- static double log1p(double *arg*) : natural log of *arg*+1
- static double pow(double *x*, double *y*): *x* to the power *y*

Rounding Functions in Math Class

- static int abs(int *arg*)
- static long abs(long *arg*)
- static float abs(float *arg*)
- static double abs(double *arg*)
- static double ceil(double *arg*)
- static double floor(double *arg*)
- static double rint(double *arg*)
- static int round(float *arg*)
- static long round(double *arg*)

Miscellaneous Functions in Math Class

- static double hypot(double *side1*, double *side2*)
- static double random()
- static float signum(double *arg*)
- static float signum(float *arg*)
- static double toDegrees(double *angleInRadians*)
- static double toRadians(double *angleInDegrees*)

StrictMath Class

- **StrictMath** class defines complete set of methods as in **Math**.
- **StrictMath** version is guaranteed to generate precisely identical results across all Java implementations but methods in **Math** are given more latitude in order to improve performance.

Package Class

- **Package** class encapsulates version data associated with a package.
- Some methods of **Package** class are:
  - String getName()
    - Returns the name of the invoking object.
  - static Package getPackage(String *pkgName*)
    - Returns a **Package** object with the name specified by *pkgName*.
  - static Package[] getPackages()
    - Return all package about which the invoking object is aware.
  - String getSpecificationTitle()
    - Returns the title of the invoking package's specification.

Methods in Package Class

- String getSpecificationVendor()
  - Returns the name of the owner of the specification for the invoking package.
- String getSpecificationVersion()
  - Returns the invoking package's specification version number.
- String toString()
  - Returns the string equivalent of the invoking package.

Enum Class

- An enumeration is list of named constants created by using keyword **enum**.
- All enumerations automatically inherit **Enum** class.
- **Enum** is a generic class declared as:
  - class Enum<E extends Enum<E>>
    - Here, **E** stands for the enumeration type.
- **Enum** has no public constructors. The only constructor is protected:
  - protected Enum(String *name*, int *ordinal*)
    - *name* is name of enum constant and *ordinal* is ordinal of this enum constant.

CharSequence Interface

- The **CharSequence** interface defines methods that grant read-only access to a sequence of characters.
- This interface is implemented by **String**, **StringBuffer**, and **StringBuilder** classes.
- Some methods of **CharSequence** interface are:
  - char charAt(int *index*)
  - int length()
  - CharSequence subSequence(int *startIndex*, int *endIndex*)
  - String toString()

Cloneable Interface

- The **Cloneable** interface defines no members. It is just used to indicate that a class allows bitwise copy of an object to be made.
- Only classes that implement the **Cloneable** interface can be cloned using popular method **clone()** declared as protected.

Comparable Interface

- Objects of classes which implement **Comparable** interface can be ordered.
- **Comparable** is generic and is declared like:
  - interface Comparable<T>
    - Here, **T** represents the type of object being compared.
- One method used for ***natural ordering*** is:
  - int compareTo(T *object*)
- The **Comparable** interface is implemented by **Byte**, **Character**, **Double**, **Float**, **Long**, **Short**, **String**, and **Integer** and **Enum**.

Appendable Interface

- Objects of a class that implements **Appendable** can have a character or character sequences appended to it.
- **Appendable** defines three methods:
  - Appendable append(char *ch*) throws IOException
  - Appendable append(CharSequence *chars*) throws IOException
  - Appendable append(CharSequence *chars*, int *begin*, int *end*) throws IOException

Iterable Interface

- **Iterable** must be implemeted by any class whose objects will be used by for-each loop.
- **Iterable** is a generic interface with declaration as:
  - interface Iterable<T>
    - Here, T is the type of object being iterated.
- One method **iterator()** is also defined in the **Iterable** interface as:
  - Iterator<T> iterator()

java.lang.reflect.Constructor

- **Constructor** is a final class which extends **AccessibleObject** class and implements **Member** and **GenericDeclaration** interface.
- It provides information about, and access to. A single constructor for a class.

java.lang.reflect.Constructor

- Some of the important methods in **Constructor** class are:
  - Class<T> getDeclaringClass()
  - Type[] getGenericParameterTypes()
  - String getName()
  - T newInstance(Object... *initArgs*)
  - String toGenericString()
  - String toString()
  - boolean isVarArgs()

java.lang.reflect.Method

- **Method** class extends **AccessibleObject** and implements **GenericDeclaration** and **Member** interface.
- It provides information about, and access to, a single method on a class or interface. The reflected method may be a class method or instance method (including an abstract method).

java.lang.reflect.Method

- Some of the important methods in **Constructor** class are:
  - Class<?> getDeclaringClass()
  - Class<?>[] getExceptionTypes()
  - Class<?>[] getParameterTypes()
  - Class<?> getReturnType()
  - String getName()
  - Object incvoke(Object *object*, Object… *args*)
  - String toGenericString()
  - String toString()
  - boolean isVarArgs()

java.lang.reflect.Field

- **Field** class extends **AccessibleObject** class and implements **Member** interface.
- It provides information about, and dynamic access to, a single field of a class or an interface. The reflected field may be a class (static) field or an instance field.

java.lang.reflect.Field

- Some of the important methods in **Constructor** class are:
  - Object get(Object *object*)
  - boolean getBoolean(Object *object*)
  - byte getByte(Object *object*)
  - char getChar(Object *object*)
  - double getDouble(Object *object*)
  - float getFloat(Object *object*)
  - int getInt(Object *object*)
  - long getLong(Object *object*)
  - short getShort(Object *object*)
  - Class<T> getDeclaringClass()

java.lang.reflect.Field

- Class<?> getType()
- String getName()
- void set(Object *object*, Object *value*)
- void setBoolean(Object *object*, boolean *b*)
- void setChar(Object *object*, char *c*)
- void setInt(Object *object*, int *i*)
- String toGenericString()
- String toString()