

UNIT 9

Understanding Core Packages

(3 Hrs.) — Object Oriented Programming in Java
Course Code: CACS204 | Year/Semester: II/III

Syllabus Coverage: Using the java.lang package; java.lang.Math; wrapper classes and associated methods; Number, Double, Float, Integer, Byte, Short, Long, Character and Boolean; using java.util core classes including Vector, Stack, Dictionary, Hashtable, Enumerations and random number generation.

UNIT ROADMAP

- Understand the role of the java.lang and java.util packages.
- Use common Math methods for numerical operations.
- Understand primitive wrapper classes and their common methods.
- Work with Vector, Stack, Dictionary and Hashtable.
- Understand Enumeration and generate random numbers.

1. Core Packages in Java

A package is a collection of related Java classes and interfaces. Java provides many standard packages so programmers can reuse tested functionality instead of implementing everything from scratch. Two important packages in this unit are java.lang and java.util.

Package	Main Role	Examples
java.lang	Fundamental language and runtime classes	Math, Integer, Double, Character, Boolean
java.util	Utility and collection-related classes	Vector, Stack, Hashtable, Random, Enumeration

Classes in java.lang are automatically available to every Java program, so an explicit import is normally not required for classes such as Math, Integer and String.

2. Using the java.lang Package

The java.lang package contains classes fundamental to Java programming. It includes classes and interfaces related to strings, mathematical operations, primitive wrappers, threads, exceptions and the basic Object hierarchy.

Important Class	Purpose
Object	Root class of the Java class hierarchy
String	Represents sequences of characters
Math	Provides common mathematical methods and constants
Integer	Wrapper class for int
Double	Wrapper class for double
Float	Wrapper class for float
Byte	Wrapper class for byte
Short	Wrapper class for short
Long	Wrapper class for long
Character	Wrapper/helper class for char
Boolean	Wrapper class for boolean

3. java.lang.Math

The Math class provides static methods for common mathematical calculations. Because its methods are static, they can be called directly using the class name.

Method	Purpose	Example
Math.abs(x)	Absolute value	Math.abs(-8) → 8
Math.max(a,b)	Larger value	Math.max(5, 9) → 9
Math.min(a,b)	Smaller value	Math.min(5, 9) → 5
Math.sqrt(x)	Square root	Math.sqrt(25) → 5.0
Math.pow(a,b)	a raised to power b	Math.pow(2, 3) → 8.0
Math.ceil(x)	Smallest integer-valued double $\geq x$	Math.ceil(4.2) → 5.0
Math.floor(x)	Largest integer-valued double $\leq x$	Math.floor(4.8) → 4.0
Math.round(x)	Rounds to nearest integer value	Math.round(4.6) → 5
Math.random()	Returns a pseudorandom double in [0.0, 1.0)	Math.random()

```

public class MathDemo {
    public static void main(String[] args) {
        System.out.println(Math.sqrt(49));
        System.out.println(Math.pow(2, 4));
        System.out.println(Math.max(10, 25));
        System.out.println(Math.round(7.6));
    }
}

```

Generating a Random Integer from 1 to 10

```
int n = (int)(Math.random() * 10) + 1;
```

Math.random() gives a value from 0.0 up to, but not including, 1.0.

Multiplying by 10 and converting to int gives 0 through 9; adding 1 changes the range to 1 through 10.

4. Wrapper Classes

Java provides wrapper classes that represent primitive values as objects. The main primitive-to-wrapper relationships are shown below.

Primitive	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Wrapper classes are useful when an object is required, such as in many collection APIs. They also provide useful conversion and comparison methods.

5. Number Class

Number is an abstract class in java.lang and is the superclass of several numeric wrapper classes, including Byte, Short, Integer, Long, Float and Double.

Method	Purpose
byteValue()	Converts the numeric value to byte
shortValue()	Converts the numeric value to short
intValue()	Converts the numeric value to int
longValue()	Converts the numeric value to long
floatValue()	Converts the numeric value to float
doubleValue()	Converts the numeric value to double

```
Integer x = 25;
```

```
System.out.println(x.intValue());
```

```
System.out.println(x.doubleValue());
```

```
System.out.println(x.longValue());
```

6. Double and Float

Double wraps a double value and Float wraps a float value. Both extend Number and provide parsing and conversion methods.

```
Double d = Double.valueOf("12.5");  
double x = Double.parseDouble("45.75");
```

```
Float f = Float.valueOf("3.5");  
float y = Float.parseFloat("8.25");
```

```
System.out.println(d);  
System.out.println(x);  
System.out.println(f);  
System.out.println(y);
```

Class	Useful Methods / Members	Purpose
Double	parseDouble(String)	Converts String to primitive double
Double	valueOf(String)	Creates a Double object
Double	isNaN()	Checks for NaN
Double	isInfinite()	Checks for infinity
Float	parseFloat(String)	Converts String to primitive float
Float	valueOf(String)	Creates a Float object
Float	isNaN()	Checks for NaN
Float	isInfinite()	Checks for infinity

7. Integer, Byte, Short and Long

These wrapper classes correspond to the four integral primitive types. They provide conversion, parsing, comparison and useful constants.

Class	Primitive	Common Methods
Integer	int	parseInt(), valueOf(), intValue(), compareTo()
Byte	byte	parseByte(), valueOf(), byteValue(), compareTo()
Short	short	parseShort(), valueOf(), shortValue(), compareTo()
Long	long	parseLong(), valueOf(), longValue(), compareTo()

```
int a = Integer.parseInt("100");  
long b = Long.parseLong("5000");
```

```
short c = Short.parseShort("25");
byte d = Byte.parseByte("10");

System.out.println(a + b + c + d);
```

Parsing Strings

Integer.parseInt("123") returns primitive int 123.
 Long.parseLong("123456") returns primitive long.
 valueOf("123") returns a wrapper object rather than a primitive.

8. Character Class

The Character class wraps a char value and provides methods for testing and converting characters.

Method	Purpose	Example
isLetter(c)	Checks whether c is a letter	Character.isLetter('A')
isDigit(c)	Checks whether c is a digit	Character.isDigit('7')
isWhitespace(c)	Checks for whitespace	Character.isWhitespace(' ')
isUpperCase(c)	Checks uppercase	Character.isUpperCase('A')
isLowerCase(c)	Checks lowercase	Character.isLowerCase('a')
toUpperCase(c)	Converts to uppercase	Character.toUpperCase('b')
toLowerCase(c)	Converts to lowercase	Character.toLowerCase('B')

```
char ch = 'A';

System.out.println(Character.isLetter(ch));
System.out.println(Character.isDigit(ch));
System.out.println(Character.toLowerCase(ch));
```

9. Boolean Class

The Boolean class wraps a boolean value and provides conversion and comparison operations.

Method	Purpose
parseBoolean(String)	Converts a String to primitive boolean
valueOf(boolean)	Creates a Boolean object
booleanValue()	Returns the primitive boolean value
compareTo(Boolean)	Compares two Boolean objects

```
boolean a = Boolean.parseBoolean("true");
Boolean b = Boolean.valueOf(false);

System.out.println(a);
System.out.println(b.booleanValue());
```

Boolean.parseBoolean() returns true when the supplied string represents "true" ignoring case; otherwise it returns false.

10. Autoboxing and Unboxing

Autoboxing is the automatic conversion of a primitive value into its corresponding wrapper object. Unboxing is the automatic conversion from a wrapper object back to its primitive value.

```
int x = 10;

// Autoboxing
Integer obj = x;

// Unboxing
int y = obj;

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

Operation	Meaning
Autoboxing	primitive → wrapper object
Unboxing	wrapper object → primitive

11. Using the java.util Package

The java.util package contains utility classes and collection-related classes. This unit focuses on Vector, Stack, Dictionary, Hashtable, Enumeration and random number generation.

Class / Interface	Main Purpose
Vector	Resizable array-like collection
Stack	LIFO collection based on Vector
Dictionary	Abstract key-value mapping class
Hashtable	Key-value hash table implementation
Enumeration	Legacy interface for enumerating elements
Random	Generates pseudorandom values

12. Vector

Vector is a growable array. Its capacity can increase as elements are added. It supports indexed access and several legacy-style methods.

```
import java.util.Vector;

Vector<String> names = new Vector<>();

names.add("Ram");
```

```
names.add("Sita");
names.add("Hari");
```

```
System.out.println(names);
System.out.println(names.get(1));
System.out.println(names.size());
```

Method	Purpose
add(e)	Adds an element
addElement(e)	Adds an element
get(index)	Returns element at index
set(index, e)	Replaces an element
remove(index)	Removes an element
size()	Returns number of elements
capacity()	Returns current capacity
contains(o)	Checks whether element exists

13. Stack

Stack represents a Last-In, First-Out (LIFO) structure. The most recently added item is removed first.

```
import java.util.Stack;
```

```
Stack<Integer> stack = new Stack<>();
```

```
stack.push(10);
stack.push(20);
stack.push(30);
```

```
System.out.println(stack.peek()); // 30
System.out.println(stack.pop()); // 30
System.out.println(stack.pop()); // 20
```

Method	Purpose
push(e)	Adds an item to the top
pop()	Removes and returns the top item
peek()	Returns the top item without removing it
empty()	Checks whether stack is empty
search(o)	Searches for an item

LIFO CONCEPT

Push: add an item to the top.

Pop: remove the top item.

Peek: inspect the top item without removing it.

Therefore, the last element pushed is the first element popped.

14. Dictionary

Dictionary is an abstract class that represents a collection of key-value pairs. A key is used to locate its associated value. Hashtable is a concrete class historically derived from Dictionary.

```
import java.util.Dictionary;  
import java.util.Hashtable;
```

```
Dictionary<Integer, String> d = new Hashtable<>();
```

```
d.put(1, "Ram");  
d.put(2, "Sita");
```

```
System.out.println(d.get(1));
```

Method	Purpose
put(key, value)	Associates a value with a key
get(key)	Returns value associated with key
remove(key)	Removes key-value mapping
isEmpty()	Checks whether dictionary is empty
size()	Returns number of mappings
keys()	Returns an Enumeration of keys
elements()	Returns an Enumeration of values

15. Hashtable

Hashtable stores key-value pairs using hashing. It is a legacy collection class and does not permit null keys or null values.

```
import java.util.Hashtable;
```

```
Hashtable<Integer, String> table = new Hashtable<>();
```

```
table.put(101, "Java");  
table.put(102, "OOP");  
table.put(103, "I/O");
```

```
System.out.println(table.get(102));  
System.out.println(table.containsKey(101));  
System.out.println(table.size());
```

A Hashtable associates each key with a value. Keys should be unique within the table; inserting a value with an existing key replaces the previous value associated with that key.

16. Enumeration

Enumeration is a legacy interface used to traverse elements of certain older collection classes such as Vector and Hashtable.

```
import java.util.Vector;
import java.util.Enumeration;

Vector<String> v = new Vector<>();
v.add("A");
v.add("B");
v.add("C");

Enumeration<String> e = v.elements();

while (e.hasMoreElements()) {
    System.out.println(e.nextElement());
}
```

Method	Purpose
hasMoreElements()	Checks whether more elements remain
nextElement()	Returns the next element

Enumeration is mainly associated with legacy collection APIs. Modern Java programs commonly use the Collection framework's Iterator or enhanced for loop when those alternatives are available.

17. Random Number Generation

The Random class in java.util generates pseudorandom values of different primitive types. It can generate integers, longs, floats, doubles and booleans.

```
import java.util.Random;

Random r = new Random();

int a = r.nextInt(100); // 0 to 99
double b = r.nextDouble(); // 0.0 to less than 1.0
boolean c = r.nextBoolean();

System.out.println(a);
System.out.println(b);
System.out.println(c);
```

Method	Returns
nextInt()	A pseudorandom int
nextInt(bound)	An int from 0 inclusive to bound exclusive
nextLong()	A pseudorandom long

Method	Returns
nextFloat()	A float from 0.0 inclusive to less than 1.0
nextDouble()	A double from 0.0 inclusive to less than 1.0
nextBoolean()	A pseudorandom true or false

Random Number from 1 to 50

Random r = new Random();

int n = r.nextInt(50) + 1;

nextInt(50) gives values from 0 through 49; adding 1 changes the range to 1 through 50.

18. Math.random() vs Random

Feature	Math.random()	java.util.Random
Package	java.lang	java.util
Typical result	double	Multiple types
Range of basic method	0.0 inclusive to 1.0 exclusive	Depends on method
Example	Math.random()	r.nextInt(100)
Best use	Simple random double	More flexible random generation

19. Combined Example

```
import java.util.Random;
import java.util.Vector;
import java.util.Stack;
```

```
public class CorePackageDemo {
    public static void main(String[] args) {
```

```
        Random random = new Random();
```

```
        Vector<Integer> numbers = new Vector<>();
        for (int i = 0; i < 5; i++) {
            numbers.add(random.nextInt(100));
        }
```

```
        System.out.println("Vector: " + numbers);
```

```
        Stack<Integer> stack = new Stack<>();
        stack.push(10);
        stack.push(20);
        stack.push(30);
```

```
        System.out.println("Top: " + stack.peek());
        System.out.println("Removed: " + stack.pop());
```

```
        System.out.println("Square root: " + Math.sqrt(144));
```

```

        System.out.println("Maximum: " + Math.max(25, 40));
    }
}

```

What This Example Demonstrates

Random generates values.

Vector stores a dynamic sequence of values.

Stack demonstrates LIFO behavior.

Math provides static mathematical methods.

All these classes reduce the need to implement common functionality manually.

20. Important Exam Comparison

Topic	Remember
Math	Static mathematical utility methods
Number	Abstract superclass of numeric wrappers
Integer	Wrapper for int
Double	Wrapper for double
Float	Wrapper for float
Long	Wrapper for long
Short	Wrapper for short
Byte	Wrapper for byte
Character	Wrapper/helper class for char
Boolean	Wrapper for boolean
Vector	Growable array-like collection
Stack	LIFO structure
Dictionary	Abstract key-value mapping
Hashtable	Hash-based key-value collection
Enumeration	Legacy traversal interface
Random	Pseudorandom number generation

KEY TAKEAWAYS

`java.lang` contains fundamental classes and is automatically imported.

`Math` provides static methods such as `abs()`, `max()`, `min()`, `sqrt()`, `pow()`, `ceil()`, `floor()` and `round()`.

Wrapper classes represent primitive values as objects and provide useful conversion methods.

`Number` is the superclass of major numeric wrapper classes.

`Character` provides useful character-testing and conversion methods.

`Boolean` provides boolean parsing and conversion operations.

`Vector` is a growable array-like collection.

`Stack` follows LIFO order using `push()`, `pop()` and `peek()`.

`Dictionary` represents key-value mappings; `Hashtable` is a concrete legacy implementation.

`Enumeration` traverses elements using `hasMoreElements()` and `nextElement()`.

`Random` provides pseudorandom values such as integers, doubles and booleans.

21. Quick Revision Questions

- What is a package in Java?
- Why is `java.lang` important?
- Explain any five methods of the `Math` class.
- What are wrapper classes? List primitive and wrapper pairs.
- What is the `Number` class?
- Differentiate `parseInt()` and `valueOf()`.
- Explain important methods of `Character`.
- What are autoboxing and unboxing?
- What is `Vector`? Mention common `Vector` methods.
- Explain the LIFO principle of `Stack`.
- What is `Dictionary`?
- What is `Hashtable`?
- What is `Enumeration`? Explain `hasMoreElements()` and `nextElement()`.
- How is random number generation performed using `Random`?
- Differentiate `Math.random()` and `java.util.Random`.

22. Short Exam Answers

What is a Wrapper Class?

A wrapper class is a `java.lang` class that represents a primitive data type as an object. Examples include `Integer` for `int`, `Double` for `double`, `Character` for `char` and `Boolean` for `boolean`.

What is Stack?

Stack is a LIFO data structure. Elements are inserted using `push()`, removed using `pop()`, and inspected using `peek()`.

What is Hashtable?

Hashtable is a hash-based key-value collection in `java.util`. It maps keys to values and is a legacy class; it does not allow null keys or null values.

What is Enumeration?

Enumeration is a legacy interface used to traverse elements of older collection classes. Its principal methods are `hasMoreElements()` and `nextElement()`.

— *End of Unit 9* —

bcalink.com