Programming in Java

Dr. Nyein Aye Maung Maung
Dr. Eng (Ritsumeikan University, Japan)
Lecturer
Computer Engineering and Information Technology Dept.
Yangon Technological University

Course Schedule

Week	Topics
Week 1	Overview of JAVA, Data Types, Variables and Arrays
Week 2	Operators and Control Statements
Week 3	Classes and A closer look at Methods and Classes
Week 4	Inheritance, Polymorphism, Abstraction and Encapsulation
Week 5	Packages and Interfaces
Week 6	Exception Handling and Multi-threaded Programming
Week 7	String Handling
Week 8	Exploring java.lang and More utilities classes
Week 9	Java Collections Framework
Week 10	Java I/O
Week 11	Basic Graphical User Interface
Week 12	Event Handling
Week 13	Database Programming
Week 14	Applet and Networking

Lecture 8

Exploring java.lang & More utilities classes

Outline of Class (Lecture 8)

- Classes and interfaces defined by **java.lang** package
  - **Wrapper** classes
  - The **Object** class
  - The **Math** class
- More Utility Classes

Topic 1

java.lang Package

java.lang Package

- It is a core package in Java.
- When classes from this package are referenced there is no need for import statement.
- Contains core set of classes such as:
  - Wrapper Classes
  - System
  - Object
  - Number
  - Math
  - Class
  - String
  - StringBuffer

1. Wrapper Classes

- Processing Primitive Data Type Values as Objects
  - A primitive type value is not an object, but it can be wrapped in an object using a wrapper class in the Java API.
  - Since many Java methods require the use of **objects as arguments**. Java offers a convenient way to incorporate, or wrap, **a primitive data type into an object**

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

1. Wrapper Classes (cont')

- **Constructor**

  Double D = new Double(3.6) //from value
  Double D=new Double("3.6") //from String value
  Integer I= new Integer(14); //from value
  Integer I=new Integer("14"); //from String value
  * The wrapper classes do not have no-arg constructors.

- **Comparing objects**
  - Obj1.compareTo(Obj2);

  // Obj1=Obj2 → 0, Obj1<Obj2 → -1, Obj1>Obj2 → 1

  D.compareTo(new Double(6.5)); → -1 ,
  I.compareTo(new Integer(12)); → 1

Encoding/Decoding Primitive Objects

- Parsing/decoding **primitive objects from Strings**
  - **ObjType.valueOf(String str)**

    eg. Integer value=Integer.valueOf(“126”);
    Double value=Double.valueOf(“126.23”);

- Converting/encoding **primitive objects to Strings**
  - **Obj.toString( )**

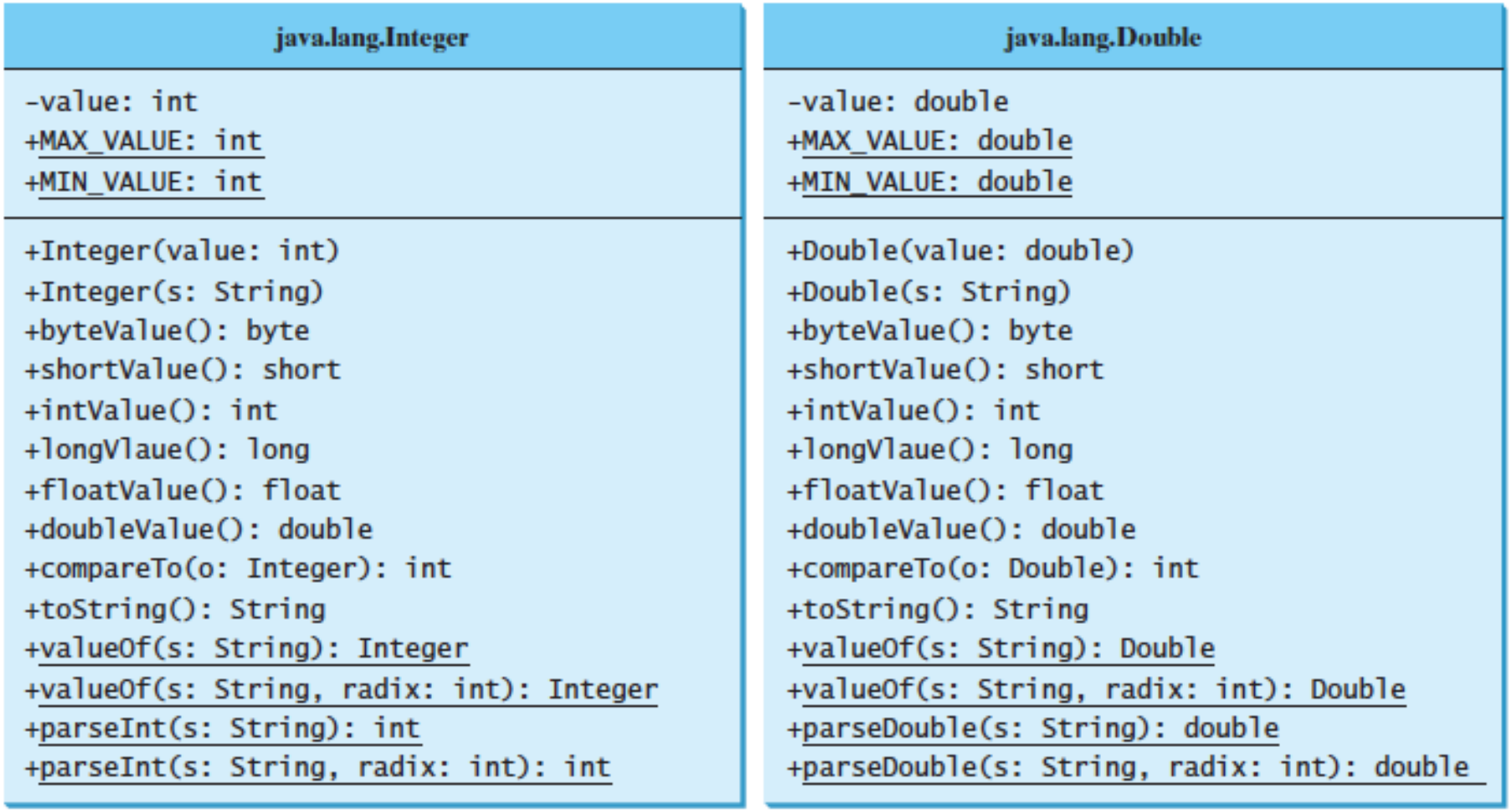
    eg. Double d=new Double(23.65);
    String s=d.toString( ); //s=“23.65”;

- Converting to **primitive data types from Objects**
  - **intValue(); longVlaue(); floatValue(); doubleValue()**

    eg double d=D.doubleValue(); int i=D.intValue();

- Converting to **primitive data types from Strings**
  - **parseInt(String str), parseLong(String str), parseFloat(String str), parseDouble(String str)**

    eg. Integer value = Integer.parseInt(“126");



Constructors, constants, and conversion methods for manipulating various data types

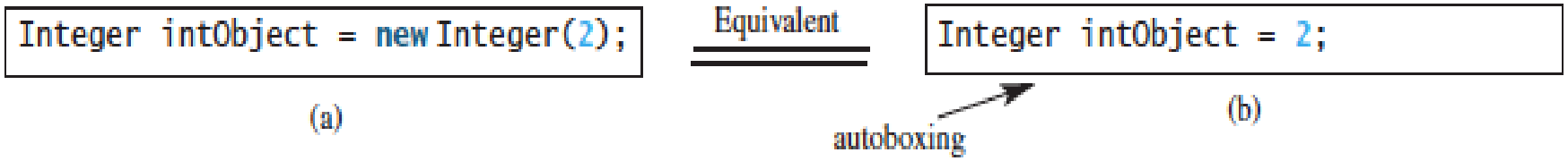
Example 1 : A program that decodes a string object to Interger object, and convert Integer value to Binary, Octal and Hexadecimal values.

```
import java.util.Scanner;
public class parse {
public static void main(String[] args) {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter a number: ");
        String numS=s.nextLine();
        Integer x = Integer.parseInt(numS);
        System.out.println(" Using parseInt: "+ x);
        System.out.println(x + " in binary: " +Integer.toBinaryString(x));
        System.out.println(x + " in octal: " +Integer.toOctalString(x));
        System.out.println(x + " in hexa: " +Integer.toHexString(x));
        System.out.println("Comparing x with another integer:"+
         x.compareTo(new Integer(4)));
}
}
```

```
Enter a number: 45
 Using parseInt: 45
45 in binary: 101101
45 in octal: 55
45 in hexa: 2d
Comparing x with another integer:1
```

AutoBoxing and AutoUnboxing

- Boxing → Converting a primitive value to a wrapper object
- Unboxing → Converting a wrapper object to a primitive value
  - Java allows primitive types and wrapper classes to be converted automatically.
  - Autoboxing & Autounboxing → **Automatic Conversion between Primitive Types and Wrapper Class Types**




```
1  Integer[] intArray = {1, 2, 3};
2  System.out.println(intArray[0] + intArray[1] + intArray[2]);
```

Line 1: Autoboxing
Line 2: Autounboxing

Character Class

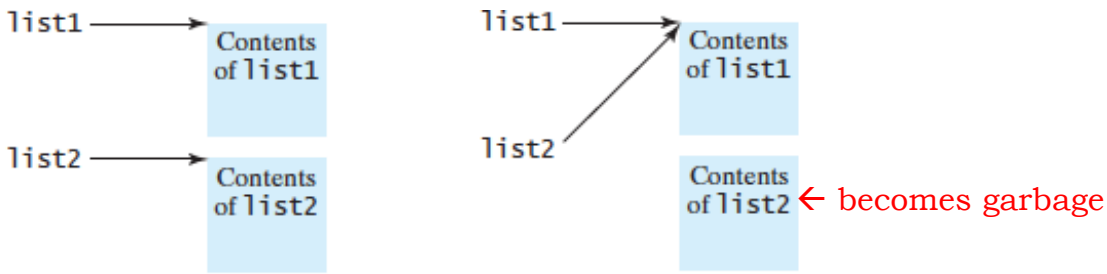
- Wrapper for primitive type ‘char’

Example 2: Character Wrapper Demo

```
class CharacterDemo {
    public static void main(String args[]) {
        char a[] = {'a', 'b', '5', '?', 'A', ' '};
        for(int i=0; i<a.length; i++) {
            if(Character.isDigit(a[i]))
            System.out.println(a[i] + " is a digit.");
            if(Character.isLetter(a[i]))
            System.out.println(a[i] + " is a letter.");
            if(Character.isWhitespace(a[i]))
            System.out.println(a[i] + " is whitespace.");
            if(Character.isUpperCase(a[i]))
            System.out.println(a[i] + " is uppercase.");
            if(Character.isLowerCase(a[i]))
            System.out.println(a[i] + " is lowercase.");
            System.out.println(a[i] + " to lowercase is ”+Character.toLowerCase(a[i]));
            System.out.println(a[i] + " to uppercase is ”+Character.toUpperCase(a[i]));
            } } }
```

2. System Class

- currentTimeMillis( )
  - return the current time in terms of milliseconds since midnight, January 1, 1970
  - useful to check how long various parts of your program take to execute
- arraycopy( )
  - To copy the contents of one array into another, you have to copy the array's individual elements into the other array.
    - eg. list2 = list1 does not copy the contents of list1, but instead copies the reference



  - arraycopy ( ) → arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);
    - src_pos and tar_pos indicate the starting positions in sourceArray and targetArray

Example 3: Write a program that displays the elapsed time for a 'for loop' from 0 to 10000.

```
class Elapsed {
    public static void main(String args[]) {
        long start, end;
        System.out.println("Timing a for loop from 0 to 1,000,0");
        start = System.currentTimeMillis(); // get starting time
        for(int i=0; i < 10000; i++) ;
        end = System.currentTimeMillis(); // get ending time
        System.out.println("Elapsed time: " + (end-start));
    }
}
```

Example 4: Array Copy Demo

```
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));
    }
    }
```

```
a = ABCDEFGHIJ
b = MMMMMMMMMM
a = ABCDEFGHIJ
b = ABCDEFGHIJ
a = AABCDEFGHI
b = BCDEFGHIJJ
```

3. Object Class

- Method **clone( )**
  - Used to make a copy of an object

```
House h1;
h1 = new House();
House h1copy =h1;
```

//This statement does not create a duplicate object.
//It simply assigns the reference of **h1** to **h1copy**.

  - To create a new object with separate memory space, use the **clone()** method

```
House h1copy =(House)h1.clone();
```

    - New instance of the class is created, but all containing fields are the same.

- **Cloneable Interface**
  - Is used to indicate that a class allows a bitwise copy of an object (i.e. clone) to be made
    - If try to call clone( ) on a class that does not implement Cloneable, a **CloneNotSupportedException** is thrown

Example 4: Demonstrate the clone( ) method.

//Implement cloneStudent class and implements Cloneable interface to allow other Objects to Clone

```
class cloneStudent implements Cloneable{
    int rollno;
    String name;
    cloneStudent(int rollno,String name){
    this.rollno=rollno;
    this.name=name;
    }
    public void setData(int rollno, String name)
    {
        this.rollno=rollno;
        this.name=name;
    }
```

```
    public static void main(String args[]){
        try{
        cloneStudent s1=new cloneStudent(101,"Mg Hla");
        cloneStudent s2=(cloneStudent) s1.clone();  //Clone s1
        System.out.println(s1.rollno+" "+s1.name);  //101 Mg Hla
        System.out.println(s2.rollno+" "+s2.name); //101 Mg Hla

        s1.setData(202, "Hla Hla");
        //Modify s1
        System.out.println(s1.rollno+" "+s1.name);  //202 Hla Hla
        //Check whether the object cloned from it (s2) has effect or not
        System.out.println(s2.rollno+" "+s2.name); //101 Mg Hla

        } catch(CloneNotSupportedException c){
                        System.out.println("Please implements Cloneable I
        nterface!!!");
                }

    }
}
```

4. Math class

- Contain methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

  a. **Trigonometric Methods**

  b. **Exponent Methods**

  c. **Rounding Methods**

  d. **min, max, and abs Methods**

  e. **The random Method**

(a) Trigonometric Methods

public static double **sin**(double radians) //angle in radian to sine
public static double **cos**(double radians) //angle radian to cos
public static double **tan**(double radians) //angle radian to tangent
public static double **toRadians**(double degree) //angle degree to radian
public static double **toDegrees**(double radians) //angle radian to degree
public static double **asin**(double a) //radian to inverse of sine
public static double **acos**(double a) //radian to inverse of cos
public static double **atan**(double a) //radian to inverse of tangent

```
Math.toDegrees(Math.PI / 2) returns 90.0
Math.toRadians(30) returns 0.5236 (same as π/6)
Math.sin(0) returns 0.0
Math.sin(Math.toRadians(270)) returns -1.0
Math.sin(Math.PI / 6) returns 0.5
Math.sin(Math.PI / 2) returns 1.0
Math.cos(0) returns 1.0
Math.cos(Math.PI / 6) returns 0.866
Math.cos(Math.PI / 2) returns 0
Math.asin(0.5) returns 0.523598333 (same as π/6)
```

(b) Exponent Methods

/** Return $\mathbf{e^x}$ */
public static double **exp**(double x)

/** Return the natural logarithm of x (**ln(x)**) */
public static double **log**(double x)

/** Return the base 10 logarithm of x (**log10(x)**) */
public static double **log10**(double x)

/** Return a raised to the power of b ($\mathbf{a^b}$) */
public static double **pow**(double a, double b)

/** Return the square root of x for x >= 0 */
public static double **sqrt**(double x)

```
Math.exp(1) returns 2.71828
Math.log(Math.E) returns 1.0
Math.log10(10) returns 1.0
Math.pow(2, 3) returns 8.0
Math.pow(3, 2) returns 9.0
Math.pow(3.5, 2.5) returns 22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5) returns 3.24
```

(c) Rounding Methods

/** x is rounded up to its nearest integer. This integer is returned as a double value. */

public static double **ceil**(double x)

/** x is rounded down to its nearest integer. */

public static double **flo**or(double x)

/** x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double. */

public static double **rint**(double x)

/** Return (int)Math.floor(x + 0.5). */

public static int **round**(float x)

/** Return (long)Math.floor(x + 0.5). */

public static long **round**(double x)

i. Math.ceil(2.1) → 3

ii. Math.ceil(2.0) → 2

iii. Math.ceil(-2.0) → -2

iv. Math.ceil(-2.1) → -2

v. Math.floor(2.1) → 2

vi. Math.floor(2.0) → 2

vii. Math.floor(-2.0) → -2

viii. Math.floor(-2.1) → -3

ix. Math.rint(2.1) → 2

x. Math.rint(-2.0) → -2

xi. Math.rint(-2.1) → -2

xii. Math.rint(2.5) → 2

xiii. Math.rint(3.5) → 4

xiv. Math.rint(-2.5) → -2

xv. Math.round(2.6f) → 3

xvi. Math.round(2.0) → 2

xvii. Math.round(-2.0f) → -2

xviii. Math.round(-2.6) → -3

xix. Math.round(-2.4) → -2

(d) min, max, and abs Methods

Math.max(2, 3) returns 3

Math.max(2.5, 3) returns 3.0

Math.min(2.5, 3.6) returns 2.5

Math.abs(-2) returns 2

Math.abs(-2.1) returns 2.1

(e) The random Method

Math.random() : returns a pseudorandom double **greater than or equal to 0.0** and **less than 1.0**

```
(int) (Math.random( ) * 10)
```
→ Returns a random integer between 0 and 9

```
50 + (int) (Math.random( ) * 50)
```
→ Returns a random integer between 50 and 99

```
a + Math.random( ) * b
```
→ Returns a random number between a and a + b, excluding a + b

// Random character between ch1 and ch2

```
(char)(ch1 + Math.random() * (ch2 - ch1 + 1))
```

Quiz on Topic 1

Quiz 1. Can each of the following statements be compiled?

a) Integer i = new Integer("23");

b) Integer i = new Integer(23);

c) Integer i = Integer.valueOf("23");

d) Double d = new Double();

e) Double d = Double.valueOf("23.45");

f) int i = (Integer.valueOf("23")).intValue();

g) double d = (Double.valueOf("23.4")).doubleValue();

h) int i = (Double.valueOf("23.4")).intValue();

i) String s = (Double.valueOf("23.4")).toString();

- Quiz 2.
  a. How do you convert an Integer into a String?
  b. How do you convert a numeric String into an Integer?
  c. How do you convert a Double number into a String?
  d. How do you convert a numeric String into a Double value?

- Quiz 3.

Show the output of the following code.

```
public class Test {
   public static void main(String[] args) {
      Integer x = new Integer(3);
      System.out.println(x.intValue());
      System.out.println(x.compareTo(new Integer(4)));
   }
}
```

Quiz 4: Evaluate the following method calls:

a. Math.sqrt(4)
b. Math.pow(2, 2)
c. Math.max(2, Math.min(3, 4))
d. Math.min(3, 4))
e. Math.rint(-2.5)
f. Math.ceil(-2.5)
g. Math.floor(-2.5)
h. Math.round(-2.5f)
i. Math.round(-2.5)
j. Math.rint(2.5)
k. Math.ceil(2.5)
l. Math.floor(2.5)
m. Math.round(2.5f)
n. Math.round(2.5)
o. Math.round(Math.abs(-2.5))

- Quiz 5

Write an expression that obtains a random integer between 34 and 55 (excluding 55) .

//Ans: 34 + (int)(Math.random() * (55 – 34))

Write an expression that obtains a random integer between 0 and 999 (including 999).

//Ans: (int)(Math.random() * 1000)

Write an expression that obtains a random number between 5.5 and 55.5 (excluding 55.5) .

//Ans: 5.5 + (Math.random() * (55.5 – 5.5))

Write an expression that obtains a random lowercase letter.

//Ans: (char)('a' + (Math.random() * ('z' – 'a' + 1))

Topic 2

More Utility Classes

1. Date Class
2. Random Class
3. Calendar Class
4. Formatter Class
5. Scanner Class

1. Date

- **Date( ) :** Initialize the object with **the current date and time**
- **Date(long millisec) :** The number of milliseconds that have elapsed since midnight, January 1, 1970.
- **boolean after(Date date):** Returns true if the invoking Date object contains a date that is later than the one specified by date. Otherwise, it returns false.
- **boolean before(Date date):** Returns true if the invoking Date object contains a date that is earlier than the one specified by date. Otherwise, it returns false.
- **int compareTo(Date date):** same → 0, earlier → negative, later → positive
- **long getTime( ):** Returns the number of milliseconds that have elapsed since January 1, 1970.

Example 5: Write a program that displays the current time and date and displays number of milliseconds since midnight, January 1 1970.

```
// Show date and time using only Date methods.
import java.util.Date;
class DateDemo {
    public static void main(String args[]) {
    // Instantiate a Date object
    Date date = new Date();
    // display time and date using toString()
    System.out.println(date);
    // Display number of milliseconds since midnight, January 1, 1970
    GMT
    long msec = date.getTime();
    System.out.println("Milliseconds since Jan. 1, 1970 GMT = " + msec);
    }
}
```

```
Mon Jul 13 08:44:54 MMT 2015
Milliseconds since Jan. 1, 1970 GMT = 1436753694396
```

2. The Random Class

- To generate a random int, long , double , float , and boolean value

java.util.Random	
+Random()	Constructs a Random object with the current time as its seed.
+Random(seed: long)	Constructs a Random object with a specified seed.
+nextInt(): int	Returns a random int value.
+nextInt(n: int): int	Returns a random int value between 0 and n (excluding n).
+nextLong(): long	Returns a random long value.
+nextDouble(): double	Returns a random double value between 0.0 and 1.0 (excluding 1.0).
+nextFloat(): float	Returns a random float value between 0.0F and 1.0F (excluding 1.0F).
+nextBoolean(): boolean	Returns a random boolean value.

Example 6 : Write a program that creates two random objects that generate the same sequences of 10 random numbers between 0 and 1000.

```
import java.util.Random;
public class RandomTest {
        public static void main(String[] args) {
                Random random1 = new Random(3);//Use seed 3
                System.out.print("From random1: ");
                for (int i = 0; i < 10; i++)
                System.out.print(random1.nextInt(1000) + " ");

 //Random objects generate the same sequence by assigning the same seed
                Random random2 = new Random(3);//Use the same seed 3
                System.out.print("\nFrom random2: ");
                for (int i = 0; i < 10; i++)
                System.out.print(random2.nextInt(1000) + " ");

        }
}
```

```
From random1: 734 660 210 581 128 202 549 564 459 961
From random2: 734 660 210 581 128 202 549 564 459 961
```

3. Calendar

- An abstract base class for extracting detailed calendar information, such as the year, month, date, hour, minute, and second

java.util.Calendar	
`#Calendar()`	Constructs a default calendar.
`+get(field: int): int`	Returns the value of the given calendar field.
`+set(field: int, value: int): void`	Sets the given calendar to the specified value.
`+set(year: int, month: int, dayOfMonth: int): void`	Sets the calendar with the specified year, month, and date. The month parameter is 0-based; that is, 0 is for January.
`+getActualMaximum(field: int): int`	Returns the maximum value that the specified calendar field could have.
`+add(field: int, amount: int): void`	Adds or subtracts the specified amount of time to the given calendar field.
`+getTime(): java.util.Date`	Returns a `Date` object representing this calendar's time value (million second offset from the UNIX epoch).
`+setTime(date: java.util.Date): void`	Sets this calendar's time with the given `Date` object.

java.util.GregorianCalendar	
+GregorianCalendar()	Constructs a GregorianCalendar for the current time.
+GregorianCalendar(year: int, month: int, dayOfMonth: int)	Constructs a GregorianCalendar for the specified year, month, and date.
+GregorianCalendar(year: int, month: int, dayOfMonth: int, hour:int, minute: int, second: int)	Constructs a GregorianCalendar for the specified year, month, date, hour, minute, and second. The month parameter is 0-based, that is, 0 is for January.

Constant	*Description*
YEAR	The year of the calendar.
MONTH	The month of the calendar, with 0 for January.
DATE	The day of the calendar.
HOUR	The hour of the calendar (12-hour notation).
HOUR_OF_DAY	The hour of the calendar (24-hour notation).
MINUTE	The minute of the calendar.
SECOND	The second of the calendar.
DAY_OF_WEEK	The day number within the week, with 1 for Sunday.
DAY_OF_MONTH	Same as DATE.
DAY_OF_YEAR	The day number in the year, with 1 for the first day of the year.
WEEK_OF_MONTH	The week number within the month, with 1 for the first week.
WEEK_OF_YEAR	The week number within the year, with 1 for the first week.
AM_PM	Indicator for AM or PM (0 for AM and 1 for PM).

Example 7: Demonstrate the use of Calendar Class

```
import java.util.*;
public class TestCalendar {
  public static void main(String[] args) {
    // Construct a Gregorian calendar for the current date and time
    Calendar calendar = new GregorianCalendar();
    System.out.println("Current time is " + new Date());
    System.out.println("YEAR: " + calendar.get(Calendar.YEAR));
    System.out.println("MONTH: " + calendar.get(Calendar.MONTH));
    System.out.println("DATE: " + calendar.get(Calendar.DATE));
    System.out.println("HOUR: " + calendar.get(Calendar.HOUR));
    System.out.println("HOUR_OF_DAY: " + calendar.get(Calendar.HOUR_OF_DAY));
    System.out.println("MINUTE: " + calendar.get(Calendar.MINUTE));
    System.out.println("SECOND: " + calendar.get(Calendar.SECOND));
    System.out.println("DAY_OF_WEEK: " + calendar.get(Calendar.DAY_OF_WEEK));
    System.out.println("DAY_OF_MONTH: " + calendar.get(Calendar.DAY_OF_MONTH));
    System.out.println("DAY_OF_YEAR: " +calendar.get(Calendar.DAY_OF_YEAR));
    System.out.println("WEEK_OF_MONTH: " +
calendar.get(Calendar.WEEK_OF_MONTH));
```

```
System.out.println("WEEK_OF_YEAR: " + calendar.get(Calendar.WEEK_OF_YEAR));
System.out.println("AM_PM: " + calendar.get(Calendar.AM_PM));

   // Construct a calendar for September 11, 2001
   Calendar calendar1 = new GregorianCalendar(2001, 8, 11);
   String[] dayNameOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday",
       "Thursday", "Friday", "Saturday"};
   System.out.println("September 11, 2001 is a " +
     dayNameOfWeek[calendar1.get(Calendar.DAY_OF_WEEK) - 1]);
}
```

```
Current time is Tue Nov 13 10:42:01 MMT 2018
YEAR: 2018
MONTH: 10
DATE: 13
HOUR: 10
HOUR_OF_DAY: 10
MINUTE: 42
SECOND: 1
DAY_OF_WEEK: 3
DAY_OF_MONTH: 13
DAY_OF_YEAR: 317
WEEK_OF_MONTH: 3
WEEK_OF_YEAR: 46
AM_PM: 0
September 11, 2001 is a Tuesday
```

4. Formatter

Specifier	*Output*	*Example*
`%b`	a boolean value	`true` or `false`
`%c`	a character	`'a'`
`%d`	a decimal integer	`200`
`%f`	a floating-point number	`45.460000`
`%e`	a number in standard scientific notation	`4.556000e+01`
`%s`	a string	`"Java is cool"`

Format Flags

Flag	Effect
–	Left justification
#	Alternate conversion format
0	Output is padded with zeros rather than spaces
space	Positive numeric output is preceded by a space
+	Positive numeric output is preceded by a + sign
,	Numeric values include grouping separators
(	Negative numeric values are enclosed within parentheses

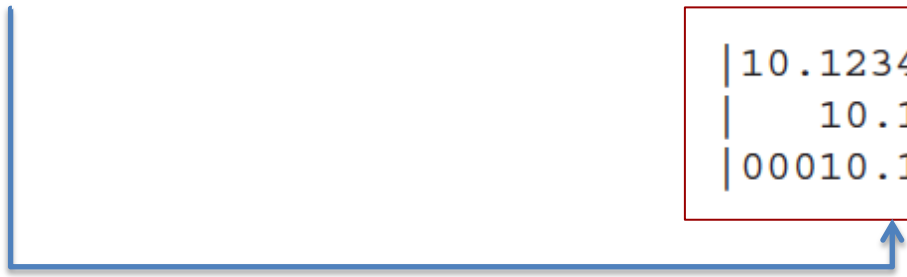
Example	*Output*
`%5c`	Output the character and add four spaces before the character item.
`%6b`	Output the boolean value and add one space before the false value and two spaces before the true value.
`%5d`	Output the integer item with width at least `5`. If the number of digits in the item is `<5`, add spaces before the number. If the number of digits in the item is `>5`, the width is automatically increased.
`%10.2f`	Output the floating-point item with width at least `10` including a decimal point and two digits after the point. Thus there are `7` digits allocated before the decimal point. If the number of digits before the decimal in the item is `<7`, add spaces before the number. If the number of digits before the decimal in the item is `>7`, the width is automatically increased.
`%10.2e`	Output the floating-point item with width at least `10` including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than `10`, add spaces before the number.
`%12s`	Output the string with width at least `12` characters. If the string item has less than `12` characters, add spaces before the string. If the string item has more than `12` characters, the width is automatically increased.

Example 8: A program that demonstrates Formatting numbers

```
import java.util.*;
class FormatDemo4 {
    public static void main(String args[]) {
        Formatter fmt = new Formatter();
        fmt.format("|%f|%n|%12f|%n|%012f|",
        10.12345, 10.12345, 10.12345);
        System.out.println(fmt);
    }
}
```

Output

```
|10.123450|
|   10.123450|
|00010.123450|
```



Example 9: A program that demonstrates formatting numbers and strings

```
import java.util.*;
public class test {
        public static void main(String[] args) {
                Formatter fmt = new Formatter();
                fmt.format("%8d%-8s\n",1234,"Java");
                System.out.print(fmt);
                fmt = new Formatter();
                fmt.format("%-8d%-8s\n",1234,"Java");
                System.out.print(fmt);
                fmt = new Formatter();
                fmt.format("%08d%-8s\n",1234,"Java");
                System.out.print(fmt);

                }
}
```

Output

```
    1234Java
1234    Java
00001234Java
```

Example 10: A program that demonstrates the use of precision.

```
import java.util.*;
public class test {
        public static void main(String[] args) {
                Formatter fmt = new Formatter();
                // Format 4 decimal places.
                fmt.format("%.4f", 123.1234567);
                System.out.println(fmt);
                // Format to 2 decimal places in a 16 character field.
                fmt = new Formatter();
                fmt.format("%016.2e", 123.1234567);
                System.out.println(fmt);
                // Display at most 15 characters in a string.
                fmt = new Formatter();
                fmt.format("%.15s", "Formatting with Java is now easy.");
                System.out.println(fmt);
}
}
```

```
123.1235
000000001.23e+02
Formatting with
```

Formatting Time and Date

- Use %t specifier with a suffix to describe the portion and precise format of the time or date desired.

a	Abbreviated weekday name
A	Full weekday name
b	Abbreviated month name
B	Full month name
c	Standard date and time string formatted as day month date hh::mm:ss tzone year
C	First two digits of year
d	Day of month as a decimal (01–31)
D	month/day/year
e	Day of month as a decimal (1–31)
F	year-month-day
h	Abbreviated month name
H	Hour (00 to 23)
I	Hour (01 to 12)
j	Day of year as a decimal (001 to 366)
k	Hour (0 to 23)
l	Hour (1 to 12)
m	Month as decimal (01 to 13)
M	Minute as decimal (00 to 59)
r	hh:mm:ss (12-hour format)
R	hh:mm (24-hour format)
T	hh:mm:ss (24-hour format)

- Example 11: A program that demonstrates formatting time and date

```
import java.util.*;
class DateTime {
public static void main(String args[]) {
Formatter fmt = new Formatter();
Calendar cal = Calendar.getInstance();
//Display standard 12-hour time format.
System.out.format("%tr", cal);
System.out.println();
//Display complete time and date information.
System.out.format("%tc", cal);
System.out.println();
//Display just hour and minute.
System.out.format("%tl:%tM", cal, cal);
System.out.println();
//Display month by name and number.
System.out.format("%tB %tb %tm", cal, cal, cal);
System.out.println();
System.out.format("%tD", cal);
System.out.println();
System.out.format("%tB %te, %tY%n", cal, cal, cal);
}
}
```

```
01:03:45 PM
Mon Jul 13 13:03:45 MMT 2015
1:03
July Jul 07
07/13/15
July 13, 2015
```

5. Scanner

boolean hasNext( )	Returns true if another token of any type is available to be read. Returns false otherwise.
boolean hasNextDouble( )	Returns true if a double value is available to be read. Returns false otherwise.
boolean hasNextFloat( )	Returns true if a float value is available to be read. Returns false otherwise.
boolean hasNextInt( )	Returns true if an int value is available to be read. Returns false otherwise.
boolean hasNextLine( )	Returns true if a line of input is available.
String next( )	Returns the next token of any type from the input source.
double nextDouble( )	Returns the next token as a double value.
float nextFloat( )	Returns the next token as a float value.
int nextInt( )	Returns the next token as an int value.
String nextLine( )	Returns the next line of input as a string.

Example 12: A program that reads double numbers from the keyboard, summing them in the process, until the user enters the string “done”. Then, it calculates the average value of the numbers and display the result.

```
class AvgNums {
      public static void main(String args[]) {
      Scanner conin = new Scanner(System.in);
      int count = 0;
      double sum = 0.0;
      System.out.println("Enter numbers to average.");
      // Read and sum numbers.
      while(conin.hasNext()) {
            if(conin.hasNextDouble()) {  //Check whether next data is Double
            sum += conin.nextDouble();
            count++;
            }
            else {
            String str = conin.next();
            if(str.equals("done")) break;  //Check whether user input equals “done”
            else {
            System.out.println("Data format error.");
            return;
            }
          }
     }
System.out.println("Average is " + sum / count); }}
```

Quiz : Which day is it today? And which day will be 2025 July 13?

```
import java.util.*;
public class Quiz {
        public static void main(String[] args) {
        Formatter fmt = new ?;
        Calendar cal=new ?;
        fmt.format("Today is %t? ', cal);
        System.out.println(fmt);
        Calendar cal2=new GregorianCalendar(?);
        fmt = new Formatter();
        fmt.format("2025 July 13 is %t? ',cal2);
        System.out.println(fmt);
}
}
```

Assignments

1. Write a program that displays (i) current date (Year, Month, Day) and time (Hour, Min, Sec) and (ii) your day of birth (eg. Friday Born).

2. Write a program that prompts user to enter day, month and year as integer and display their input with three different for mats
   1. mm/dd/yy
   2. Month dd, Year and

   Repeat the process until user enter 'No'.

3. Write a program that creates a table of squares and cubes for the numbers between 1 and 10 as shown below.

```
 1     1     1
 2     4     8
 3     9    27
 4    16    64
 5    25   125
 6    36   216
 7    49   343
 8    64   512
 9    81   729
10   100  1000
```

Next Week Lecture

- Collection Frameworks