|  |  |
| --- | --- |
| Week4 | Java lang & util packages |

**Guess The O/p**

String str=”java”;

Str.concat(“programming”);

System.out.println(str);

**NOTE**:- For substring() operation String is preffered to StringBuilder since String will create new string for substring which refers to the old string where as StringBuilder creates a new string for substring & store the values. This is upto java 7 but after in both case substring maintains a copy.

**Java.lang.Math Class**

Java.lang.Math Class methods help to perform numeric operations like square, square root, cube, cube root, exponential and trigonometric operations.

**Math is abstract class**

**all methods are static :-**

int abs(int) int/float/long/double

int min(int,int) int/float/long/double

int max(int,int) int/float/long/double

ceil() 2.x - - 3

floor() 2.x -- 2

round() 2.5 -- 3 2.4 -- 2

pow(),log(),exp(),sqrt()

sin(),cos(),tan(),asin(),acos(),atan()

toDegrees() toRadians()

random() 0.0 to 1.0

**static final datamembers:-**

PI , E

constant Math.E is Eulers’s Number, the base of natural logarithms, e, which is approximately 2.718.

The **Math.exp()** static method returns [e](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/E) raised to the power of a number. That is

𝙼𝚊𝚝𝚑.𝚎𝚡𝚙(𝚡)=ex

**INTERVIEW QUESTION**

**Random Number generation in java**  
**1) Math.random()**

**2) Random class** and

**3) ThreadLocalRandom class.**

The **Math.random()** method in the java.math  is static , thus it always return double value.The main advantage is its simplicity.

double random = Math.random();

System.out.println(random);

# Output: A random number between 0.0 (inclusive) and 1.0 (exclusive)

However, Math.random() only generates double values between 0.0 and 1.0. If you need to generate random integers or random numbers within a certain range, you’ll need to use a different approarch.

**//To generate a random number between 10 and 50**:

double random = 10 + (Math.random() \* 40);

System.out.println(random);

# Output: A random number between 10.0 (inclusive) and 50.0 (exclusive)

While Math.random() is a simple and effective way to generate random numbers in Java, there are other methods that offer more flexibility and control. Let’s explore two of them:

**Random class** in Java is used to generate pseudo-random numbers. Unlike Math.random(), the Random class can generate both integer and floating-point numbers, and it allows you to specify a range.

**//To generate a random number between 10 and 50**

import java.util.Random;

Random rand = new Random();

int random = 10+rand.nextInt(40);

System.out.println(random);

# Output: A random number between 10(inclusive) and 50 (exclusive)

The **ThreadLocalRandom** class in Java is another option for generating random numbers. It’s especially useful in multi-threaded environments, as it reduces contention and overhead associated with Random.

first import the ThreadLocalRandom class. We then use the current method to get the current thread’s ThreadLocalRandom, and call its nextInt method to generate a random integer.

**//To generate a random number between 10 and 50**

import java.util.concurrent.ThreadLocalRandom;

int random = 10+ThreadLocalRandom.current().nextInt(40);

System.out.println(random);

# Output: A random number between 10 (inclusive) and 50(exclusive)

**INTERVIEW QUESTION:-**

### **What does the string intern() method do in Java?**

If you apply the intern() method to a few strings, you will ensure that all strings having the same content share the same memory. As soon as a String object is invoked with intern(), it first checks if the string value of the String object is already present in the string pool and if it is available, then the reference to that string from the string constant pool is returned. If not, a new string object is added to the string pool, and a reference to it is returned.

**Guess the O/P**

**System.out.println(Math.min(Double.MIN\_VALUE, 0.0d));**

Ans:-0.0

**Guess the O/P**

**System.out.println(20+ +9- -12+ +4- -13+ +19);**

The statement can be written and solved as follows:(20+ (+9)- (-12)+ (+4)- (-13)+ (+19));

=20+9+12+4+13+19=77

**Guess the O/P**

(int)(3\*Math.random()) 🡺 0,1,2

**Which of the following statement runs infinitely?**

i. for( ; ; )

ii. for( ; true; )

iii. for( ; false; )

iv. for( ; 2==2; )

v. for(int i=1; i>=1; i++)

Ans:- i, ii, iv, v

**Java.lang.Enum Class**

Enum, introduced in Java 5, is a special data type that consists of a **set of pre-defined int constants named values separated by commas.** These named values are also known as **elements or enumerators or enum instances.** Since the values in the enum type are constant, you should always represent them in **UPPERCASE** letters.

The following characteristics make enum a ‘special’ class:

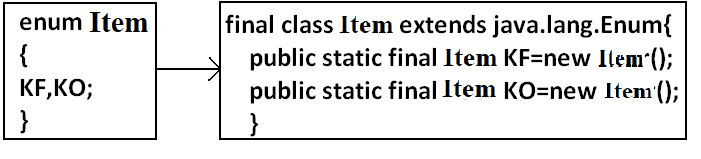
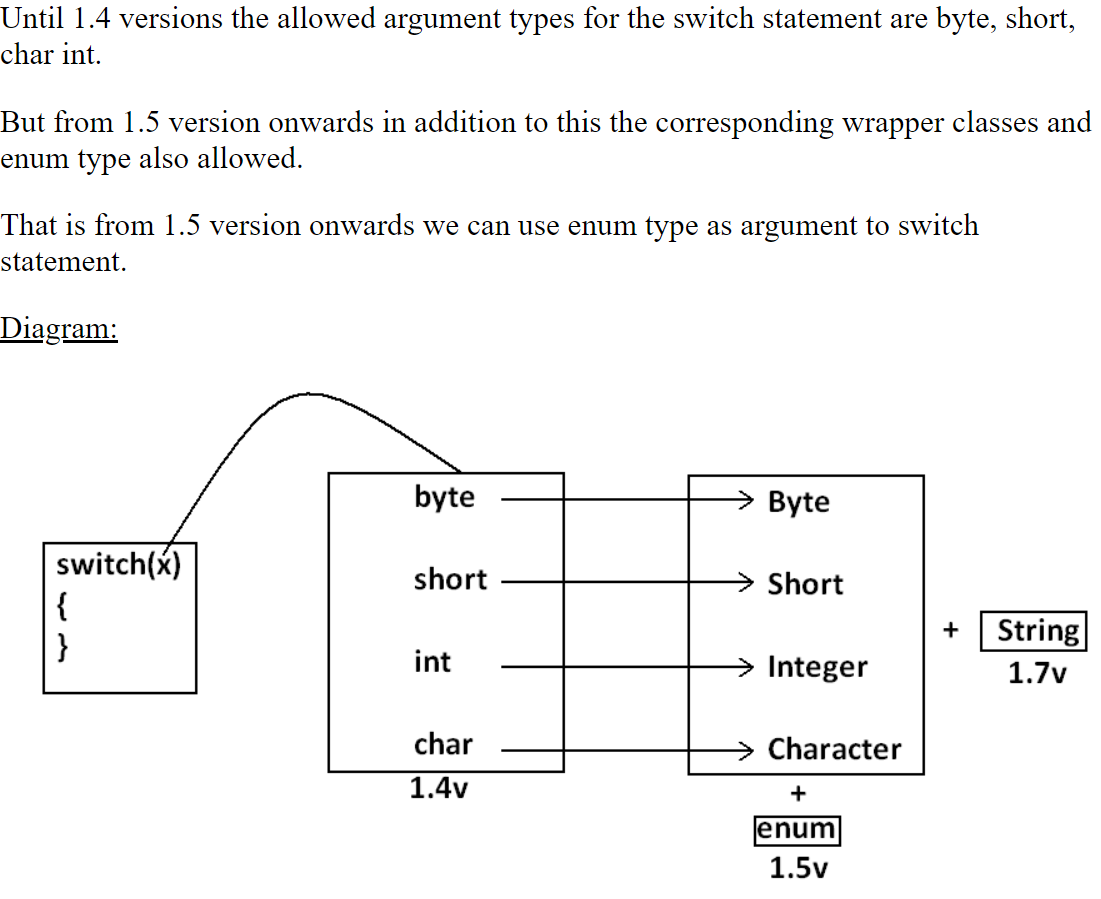
* enum constants cannot be overridden
* enum doesn’t support the creation of objects
* enum can’t extend other classes
* enum can implement interfaces like classes

to define an Enum type, the ‘enum’ keyword is used

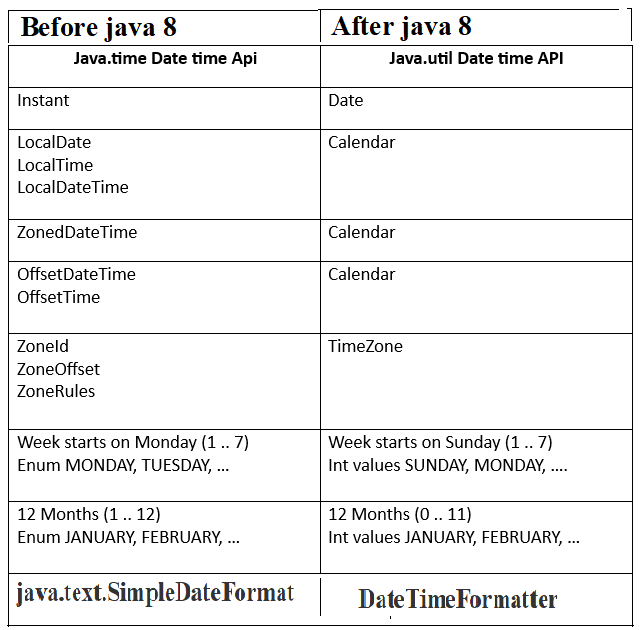
enum Variable\_Name { VALUE\_1, VALUE\_2, VALUE\_3, … }

### Internal implementation of enum:

Internally enum's are implemented by using class concept.  
Every enum constant is a reference variable to that enum type object.  
Every enum constant is implicitly**public static final** always.

Example 3:  
  
  
  


# Java Date and Time



**java.util.Date Class**

Date Class represents a specific point in time, in milliseconds.. It provides methods to manipulate and format dates, as well as perform arithmetic and comparison operations. Internally the class uses a **base or reference date of January 1, 1970, 00:00:00 GMT (also known as the Unix epoch)**.

Date(): Creates an object representing the current date and time.

Date(long millis): Creates a Date object with the specified number of milliseconds since

January 1, 1970, 00:00:00 GMT (the Unix epoch).

Date(y,m,d) 1900+ year

month 0 to 11

Date(y,m,d,h,min,s)

• toString(): Returns a string representation. The default format is not very

readable or localized.

• SimpleDateFormat class (from java.text package): Allows formatting and

parsing of dates using patterns.

boolean before(Date);

boolean after(Date);

boolean equals(Date);

int compareTo(Date) returns +1,-1,0

Object clone();

**java.util.Calendar abstract class:-**

public static Calendar getInstance();

String get(int which);

void add(int which,int value);

void set(int which,int value);

void set(y,m,d) month 0 to 11

void set(y,m,d,h,m,s,[ms]);

void clear();

void clear(int which);

before(), after(), clone(),equals()

**static final data members :-**

DATE,YEAR,MONTH

HOUR (12),MINUTE,SECOND

HOUR\_OF\_DAY (24)

MILLISECOND

DAY\_OF\_YEAR (366)

WEEK\_OF\_MONTH (4/5)

WEEK\_OF\_YEAR (52)

DAY\_OF\_WEEK [ 1 - 7]

AM\_PM 0 / 1

**GregorianCalendar class is subclass of Calendar**

GregorianCalendar()

GregorianCalendar(Locale)

GregorianCalendar(TimeZone)

GregorianCalendar(TimeZone,Locale)

public boolean isLeapYear(int year);

**java.util.Locale class**

**static final datamembers:-**

UK,US,CANADA,TAIWAN,ITALY,JAPAN,

FRENCH,...

public static Locale getDefault();

public static void setDefault(Locale);

public String getDisplayCountry();

public String getDisplayLanguage();

public String getDisplayName();

**java.util.TimeZone class**

public static TimeZone getDefault();

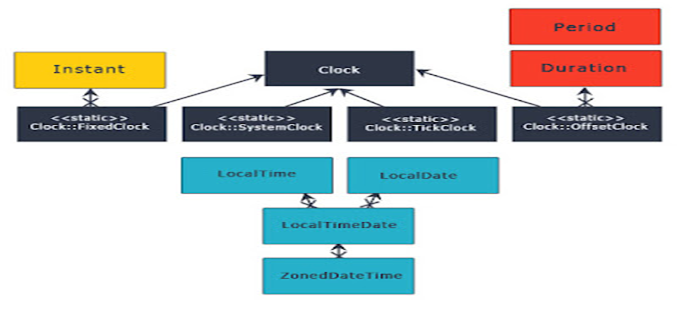
public static void setDefault(TimeZone);

public static TimeZone getTimeZone(String id);

public String getID();

public void setID(String);

public static String[] getAvailableIDs()



**Java.time.LocalDate Class**

immutable object

 It doesn’t carry any information about the offset or time zone. It stores the date in YYYY-MM-DD format, for example ‘2014-03-18’.

LocalDate.of(2015, 03, 18); -- When individual values know

LocalDate.parse("2015-03-18"); -- Creating from date string

LocalDate.now(); -- To get the current date.

LocalDate.now(ZoneId.of("America/Chicago"));

LocalDate date = LocalDate.now();

date.getMonth();

date.getDayOfYear();

date.get(ChronoField.YEAR);

**LocalTime**

Similar to LocalDate class, LocalTime represents only time of the day. It also doesn’t hold time zone details. It stores the time in HH:mm:ss.nano\_seconds format

LocalTime.of(4, 30, 15);

LocalTime.parse("04:30:15.12345");

LocalTime.now();

LocalTime.now(ZoneId.of("America/Chicago"));

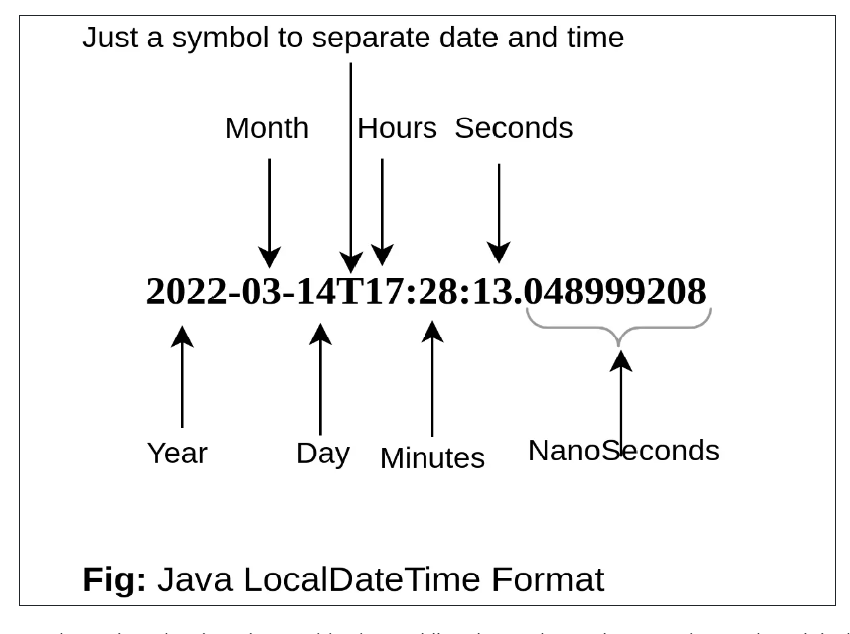
date.getMinute();

date.getNano();

date.get(ChronoField.HOUR\_OF\_DAY);

**LocalDateTime**

LocalDateTime is the combination of LocalDate and LocalTime that holds both date and time parts with out time zone details. The format of stored data is 2007-12-03T10:15:30 whete ‘T’ is the delimiter between date and time values.



LocalDateTime.now();

LocalDateTime.getDayOfWeek();

LocalDateTime.parse("2007-12-03T10:15:30");

date.toLocalDate();

date.toLocalTime();

**Instant**

Instant is a point on a continuous time line or scale. Basically this represents the number of seconds passed since the Epoch time 1970-01-01T00:00:00Z. Internally Instant stores two values, one long value representing epoch-seconds and an int representing nanosecond-of-second, which will always be between 0 and 999,999,999. Any date-time after 1970-01-01T00:00:00Z will return positive value and before will be negative value.

1. Instant.now();

2. Instant.now().getEpochSecond();

3. Instant.parse("1969-01-01T00:00:00.00Z").getEpochSecond(); --> -31,536,000

4. Instant.parse("1971-01-01T00:00:00.00Z").getEpochSecond(); --> 31,536,000

**DateTimeFormatter:** This class is the replacement for java.text.DateFormat which provides two main methods; format(temporal) to convert temporal object to string and parse(string) to create a temporal object from the given date string. Creating DateTimeFormatter instance is easy, it provides overloaded **ofpattern** methods to create it instances.

DateTimeFormatter f1 = DateTimeFormatter.ofPattern("dd-MMM-yyyy");

LocalDate date = f1.parse("18-Mar-2014");

f1.format(LocalDate.of(2014, 3, 18)); => 18-Mar-2014

//For localization

DateTimeFormatter f2= DateTimeFormatter.ofPattern("dd-MMM-yyyy", Locale.FRENCH);

f2.format(LocalDate.of(2014, 3, 18)); => 18-mars-2014

DateTimeFormatter class also contains many of its own instances like ISO\_LOCAL\_DATE, ISO\_LOCAL\_DATE\_TIME, BASIC\_ISO\_DATE etc that can be used for our gene

A temporal field is a field of date-time, such as month-of-year or hour-of-minute. These fields are represented by the TemporalField interface and the ChronoField class implements this interface. The get() or getLong() methods of the class LocaldateTime accepts a temporal field as a parameter and gets the value of the given field in the current object.

LocalDateTime lDate = LocalDateTime.now();

int field = lDate.get(ChronoField.DAY\_OF\_MONTH);

**java.util.Arrays Class**

This is a part of the Java Collections framework and provides methods to create, access and manipulate Java arrays dynamically.

All the methods provided by the Arrays class are static in nature.

The Arrays class was introduced in Java 1.2 and the methods it contains are mostly used for manipulation of the array including searching, sorting, etc.  The arrays class provides overloaded methods for almost all the data types.

|  |  |
| --- | --- |
| class hierarchy for Arrays | static< T> List<T>asList(Object[] a)  static int binarySearch (int[] a, int fromIndex, int toIndex, int key) |

**Arrays class**

all methods are static

any primitive datatype array can be navigated

public static int binarySearch(int[],int value) //before search array must be sorted

public static boolean equals(int[],int[]);

public static void fill(int[],int value);

public static void fill(int[],int stindex,int endindex,int value);

public static void sort(int[]);

public static void sort(int[],int stindex,int endindex);

If stindex > end index it raises **IllegalArgumentException**

**INTERVIEW QUESTION:-**

**Which sorting is used in Arrays class to sort in Java?**

**Ans:** The sort method of Arrays class in Java uses two sorting techniques. It uses **quicksort** when primitive types are used whereas when objects are used that implement comparable interface, **merge sort** is used.

**SORTING OF ARRAYS:-**

1. **Arrays.sort()**
2. **Collections.sort()**
3. **Comparator interface**
4. **Comparable interface**

**Arrays.sort(),** which is limited to arrays

**Collections.sort()** can sort all List implementations

The **Comparator interface** is used to order objects of an arbitrary [class](https://www.codecademy.com/resources/docs/java/classes). It is not to be confused with the [Comparable](https://www.codecademy.com/resources/docs/java/comparable) interface, which is implemented by the class to be sorted. The Comparator interface is implemented in a separate class.

Syntax

class MyComparator implements Comparator<MyClass> {

@Override public int compare(MyClass a, MyClass b)

{

// Compare logic

...

return result;

}

}

The **Comparable interface** is used to define how a [class](https://www.codecademy.com/resources/docs/java/classes) is to be sorted. It is not to be confused with the [Comparator](https://www.codecademy.com/resources/docs/java/comparator) interface, which is implemented in a separate class. The Comparable interface is implemented in the class to be sorted.

Syntax

class MyClass implements Comparable<MyClass> {

// Class body.

...

@Override public int compareTo(MyClass value)

{

// Comparison Logic

...

return result;

}

}

**INTERVIEW QUESTION**

**Differentiate Comparator & Comparable**

**Java.util. StringTokenizer Class**

**StringTokenizer** class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

**Note**: The StringTokenizer class is deprecated now. It is recommended to use the split() method of the String class or the Pattern class that belongs to the java.util.regex package.

**StringTokenizer class**

StringTokenizer(String);

StringTokenizer(String,String separators);

public int countTokens();

public boolean hasMoreTokens();

public String nextToken();