Agenda

- Abstract class/method
- Interfaces
- Marker interfaces
- Date/ LocalDate/ Calender

Abstract Methods

- If implementation of a method in super-class is not possible/incomplete, then method is declared as abstract.
- Abstract method does not have definition/implementation.
- If class contains one or more abstract methods, then class must be declared as abstract. Otherwise compiler raise an error.
- The super-class abstract methods must be overridden in sub-class; otherwise sub-class should also be marked abstract.
- The abstract methods are forced to be implemented in sub-class. It ensures that sub-class will have corresponding functionality.
- The abstract method cannot be private, final, or static.
- Example: abstract methods declared in Number class are:
    - abstract int intValue();
    - abstract float floatValue();

Abstract class

- If implementation of a class is logically incomplete, then the class should be declared abstract.
- If class contains one or more abstract methods, then class must be declared as abstract.
- An abstract class can have zero or more abstract methods.
- Abstract class object cannot be created; however its reference can be created.
- Abstract class can have fields, methods, and constructor.
- Its constructor is called when sub-class object is created and initializes its (abstract class) fields.
- Example:
- java.lang.Number
- java.lang.Enum

Fragile base class problem

- If changes are done in super-class methods (signatures), then it is necessary to modify and recompile all its sub-classes. This is called as "Fragile base class problem".
- This can be overcomed by using interfaces.

Interface (Java 7 or Earlier)

- Interfaces are used to define standards/specifications.
- A standard/specification is set of rules.
- Interfaces are immutable i.e. once published interface should not be modified.

- Interfaces contains only method declarations. All methods in an interface are by default abstract and public.
- They define a "contract" that is must be followed/implemented by each sub-class.
- Interfaces enables loose coupling between the classes i.e. a class need not to be tied up with another class implementation.
- Interfaces cannot be instantiated, they can only be implemented by classes or extended by other interfaces.
- Java 7 interface can only contain public abstract methods and static final fields (constants).
- They cannot have non-static fields, non-static methods, and constructors.
- Examples:
  - java.io.Closeable / java.io.AutoCloseable
  - java.lang.Runnable
- Multiple interface inheritance is allowed in Java

```
interface Displayable {
void display();
}
interface Acceptable {
void accept();
}
class Employee implements Displayable,Acceptable{}
```

- If two interfaces have same method, then it is implemented only once in sub-class.

class vs abstract class vs interface

- class
  - Has fields, constructors, and methods
  - Can be used standalone -- create objects and invoke methods
  - Reused in sub-classes -- inheritance
  - Can invoke overridden methods in sub-class using super-class reference -- runtime polymorphism
- abstract class
  - Has fields, constructors, and methods
  - Cannot be used independently -- can't create object
  - Reused in sub-classes -- inheritance -- Inherited into sub-class and must override abstract methods
  - Can invoke overridden methods in sub-class using super-class reference -- runtime polymorphism
- interface
  - Has only method declarations
  - Cannot be used independently -- can't create object
  - Doesn't contain anything for reusing (except static final fields)

  - Used as contract/specification -- Inherited into sub-class and must override all methods
  - Can invoke overridden methods in sub-class using super-class reference -- runtime polymorphism

Marker interfaces

- Interface that doesn't contain any method declaration is called as "Marker interface".
- These interfaces are used to mark or tag certain functionalities/features in implemented class.
- In other words, they associate some information (metadata) with the class.
- Marker interfaces are used to check if a feature is enabled/allowed for the class.
- Java has a few pre-defined marker interfaces. e.g. Serializable, Cloneable, etc.
  - java.io.Serializable -- Allows JVM to convert object state into sequence of bytes.
  - java.lang.Cloneable -- Allows JVM to create copy of the class object.

Date/ LocalDate/ Calender

- Date and Calender class are in java.util package
- The class Date represents a specific instant in time, with millisecond precision.
- the formatting and parsing of date strings were not standardized it is not recommended to use
- As of JDK 1.1, the Calendar class should be used to convert between dates and time fields and the DateFormat class should be used to format and parse date strings.
- LocalDate is in java.time Package
- It is immutable and threadsafe class.

Java DateTime APIs

- DateTime APIs till Java 7

```
// java.util.Date
Date d = new Date();
System.out.println("Timestamp: " + d.getTime());
// number of milliseconds since 1-1-1970 00:00.
SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
System.out.println("Date: " + sdf.format(d));

// java.util.Date
String str = "28-09-1983";
SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
Date d = sdf.parse(str);
System.out.println(d.toString());

// java.util.Calendar
Calendar c = Calendar.getInstance();
System.out.println(c.toString());
System.out.println("Current Year: " + calendar.get(Calendar.YEAR));
System.out.println("Current Month: " + calendar.get(Calendar.MONTH));
System.out.println("Current Date: " + calendar.get(Calendar.DATE));
```

- Limitations of existing DateTime APIs

    - Thread safety
    - API design and ease of understanding
    - ZonedDate and Time

- Most commonly used java 8 onwards new classes are LocalDate, LocalTime and LocalDateTime.

- LocalDate

```
LocalDate localDate = LocalDate.now();
LocalDate tomorrow = localDate.plusDays(1);
DayOfWeek day = tomorrow.getDayOfWeek();
int date = tomorrow.getDayOfMonth();
System.out.println("Date: " +
tomorrow.format(DateTimeFormatter.ofPattern("dd-MMM-yyyy")));

//LocalDate date = LocalDate.of(1983, 09, 28);
LocalDate date = LocalDate.parse("1983-09-28");
System.out.println("Is Leap Year: " + date.isLeapYear());
```

- LocalTime

```
LocalTime now = LocalTime.now();
LocalTime nextHour = now.plus(1, ChronoUnit.HOURS);
System.out.println("Hour: " + nextHour.getHour());
System.out.println("Time: " +
nextHour.format(DateTimeFormatter.ofPattern("HH:mm")));
```

- LocalDateTime

```
LocalDateTime now = LocalDateTime.now();
LocalDateTime dt = LocalDateTime.parse("2000-01-30T06:30:00");
dt.minusHours(2);
System.out.println(dt.toString());
```

Lab Work

- Read the given assignment keep questions ready.
- Solve the assignment
- practice the exception handling and custom exception from CPP