Integrative Programming

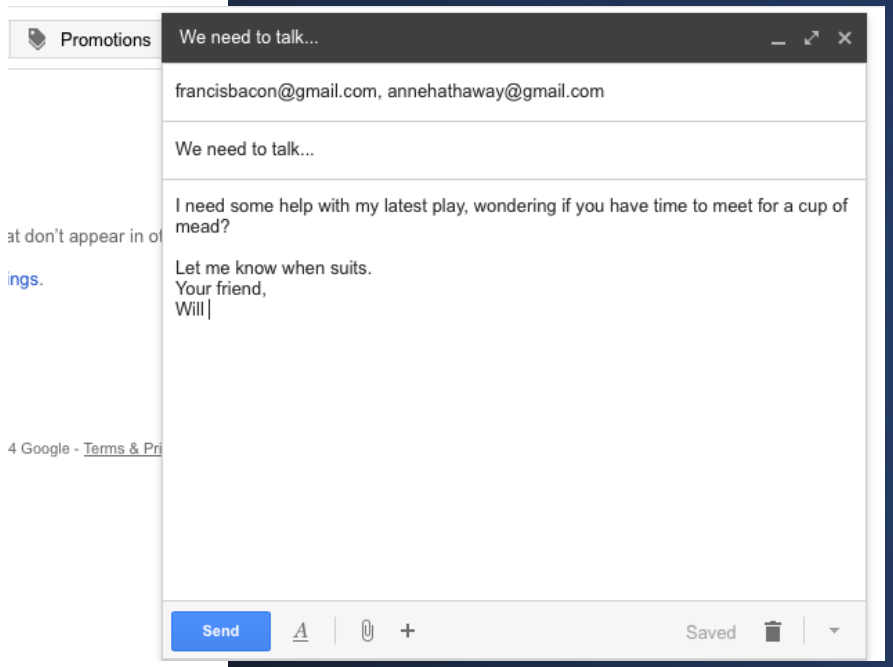
Outline

- Abstract classes
- Interface

Abstract Classes

Abstraction

- Quality vs events
- Hiding implementation
- What the objects does



Abstract Class

```java
/* File name : Employee.java */
public abstract class Employee {
   private String name;
   private String address;
   private int number;

   public Employee(String name, String address, int number) {
      System.out.println("Constructing an Employee");
      this.name = name;
      this.address = address;
      this.number = number;
   }

   public double computePay() {
     System.out.println("Inside Employee computePay");
     return 0.0;
   }

   public void mailCheck() {
      System.out.println("Mailing a check to " + this.name + " " + this.address);
   }

   public String toString() {
      return name + " " + address + " " + number;
   }

   public String getName() {
      return name;
   }

   public String getAddress() {
      return address;
   }

   public void setAddress(String newAddress) {
      address = newAddress;
   }

   public int getNumber() {
      return number;
   }
}
```

```java
/* File name : AbstractDemo.java */
public class AbstractDemo {

   public static void main(String [] args) {
      /* Following is not allowed and would raise error */
      Employee e = new Employee("George W.", "Houston, TX", 43);
      System.out.println("\n Call mailCheck using Employee reference--");
      e.mailCheck();
   }
}
```

```java
/* File name : Salary.java */
public class Salary extends Employee {
   private double salary;   // Annual salary

   public Salary(String name, String address, int number, double salary) {
      super(name, address, number);
      setSalary(salary);
   }

   public void mailCheck() {
      System.out.println("Within mailCheck of Salary class ");
      System.out.println("Mailing check to " + getName() + " with salary " + salary);
   }

   public double getSalary() {
      return salary;
   }

   public void setSalary(double newSalary) {
      if(newSalary >= 0.0) {
         salary = newSalary;
      }
   }

   public double computePay() {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
}
```

```java
/* File name : AbstractDemo.java */
public class AbstractDemo {

   public static void main(String [] args) {
      Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);
      Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);
      System.out.println("Call mailCheck using Salary reference --");
      s.mailCheck();
      System.out.println("\n Call mailCheck using Employee reference--");
      e.mailCheck();
   }
}
```

```
Constructing an Employee
Constructing an Employee
Call mailCheck using Salary reference --
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

 Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.0
```

Abstract Methods

- abstract keyword before the method name
- No body, only signature
- Semi-colon at the end instead of curly braces

```
public abstract class Employee {
    private String name;
    private String address;
    private int number;

    public abstract double computePay();
    // Remainder of class definition
}
```

Consequences of abstract method

- The class must also be declared abstract
- Child classes:
  - Override the abstract method, or
  - Declare itself as abstract

```
/* File name : Salary.java */
public class Salary extends Employee {
   private double salary;   // Annual salary

   public double computePay() {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
   // Remainder of class definition
}
```

Abstract Class Practice

Shape Class	Square Class	Circle Class	Triangle Class
• Abtract Method: getArea()			

Interface

Reasons to Use

- To achieve abstraction
- Multiple inheritance
- Achieve Loose coupling

Interface

Reference type

Similar to class

Collection of abstract methods

A class implements an interface

May contain

Constants, default methods, static methods

Comparison

Class

- Attributes
- Behaviours (methods)

Interface

- Behaviours

Similarities

- Methods
- .java
- .class
- Appear in packages

Difference of Interface vs Class

- cannot be instantiated
- No constructors
- All methods are abstract
- No instance fields
  - All fields must be declared static and final
- Implemented by a class (not extended)

Interface Declaration

```java
/* File name : Animal.java */
interface Animal {
   public void eat();
   public void travel();
}
```

Implementing Interfaces

- Class is agreeing to a contract
  - Perform (implement) the required behaviours
- Uses the implements keyword

```java
/* File name : MammalInt.java */
public class MammalInt implements Animal {

   public void eat() {
      System.out.println("Mammal eats");
   }

   public void travel() {
      System.out.println("Mammal travels");
   }

   public int noOfLegs() {
      return 0;
   }

   public static void main(String args[]) {
      MammalInt m = new MammalInt();
      m.eat();
      m.travel();
   }
}
```

Extending Interfaces

```java
// Filename: Sports.java
public interface Sports {
   public void setHomeTeam(String name);
   public void setVisitingTeam(String name);
}

// Filename: Football.java
public interface Football extends Sports {
   public void homeTeamScored(int points);
   public void visitingTeamScored(int points);
   public void endOfQuarter(int quarter);
}

// Filename: Hockey.java
public interface Hockey extends Sports {
   public void homeGoalScored();
   public void visitingGoalScored();
   public void endOfPeriod(int period);
   public void overtimePeriod(int ot);
}
```

Extending Multiple Interfaces

```
public interface Hockey extends Sports, Event
```

Practical Example

```
// To use the sqrt function
import java.lang.Math;

interface  Polygon {
   void getArea();

 // calculate the perimeter of a Polygon
   default void getPerimeter(int... sides) {
      int perimeter = 0;
      for (int side: sides) {
         perimeter += side;
      }

   System.out.println("Perimeter: " + perimeter);
   }
}
```

```
class Triangle implements Polygon {
   private int a, b, c;
   private double s, area;

// initializing sides of a triangle
   Triangle(int a, int b, int c) {
      this.a = a;
      this.b = b;
      this.c = c;
      s = 0;
   }

// calculate the area of a triangle
   public void getArea() {
      s = (double) (a + b + c)/2;
      area = Math.sqrt(s*(s-a)*(s-b)*(s-c));
      System.out.println("Area: " + area);
   }
}
```

```
class Main {
   public static void main(String[] args) {
      Triangle t1 = new Triangle(2, 3, 4);

// calls the method of the Triangle class
      t1.getArea();

// calls the method of Polygon
      t1.getPerimeter(2, 3, 4);
   }
}
```

Sources

https://www.tutorialspoint.com/java/java_abstraction.htm

https://www.slideshare.net/arulkumarcbe/interface-in-java-84442079

https://www.programiz.com/java-programming/interfaces