Welcome to the

Java Course

Module 3 – Day 04

Content of the course

- Object-Oriented Programming concepts
- Classes and objects
- Inheritance and polymorphism
- Encapsulation and accessibility
- **Interface and abstract classes**
- Exceptions

Air Flight Company Project - Step 3

- Create a parent class called Aircraft and extend it with 3 child classes called Boeing737, AirbusA320 and AirbusA380.

- The Aircraft class should allow to store the aircraft's model and capacity.

Air Flight Company Project - Step 3

- Whenever a new Flight gets created, it should no longer request the flight capacity, it should request the aircraft model instead.

- Modify the Flight class such that it no longer has a property to store the flight’s capacity but that it stores a reference to an Aircraft object instead.

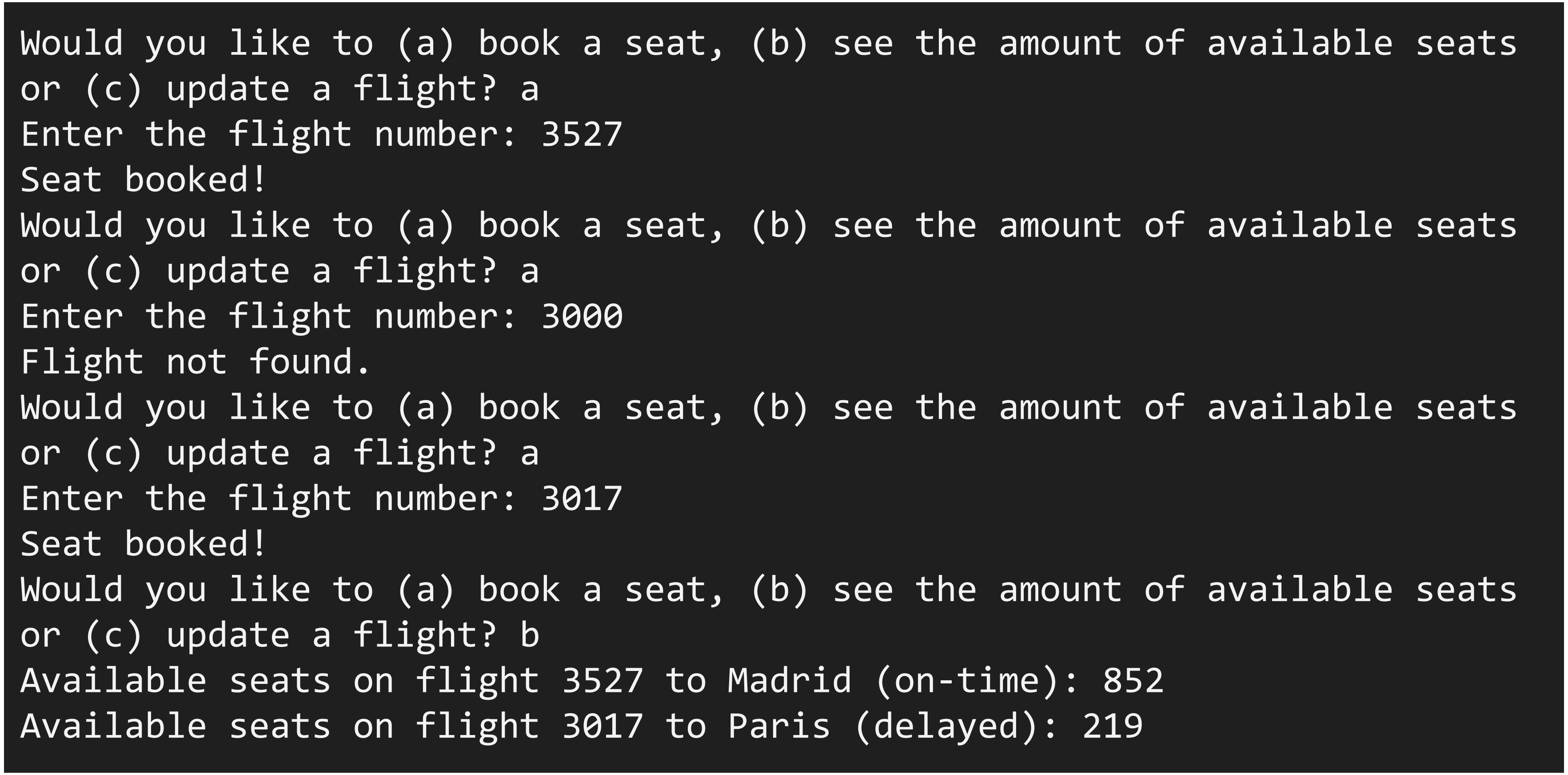
Air Flight Company Project - Step 3

```
>>> New Flight <<<
Enter flight number: 3527
Enter destination: Madrid
Enter aircraft model: AirbusA380
Flight created. Would you like to add another flight (y/n)? y

>>> New Flight <<<
Enter flight number: 3017
Enter destination: Paris
Enter aircraft model: AirbusA320
Flight created. Would you like to add another flight (y/n)? n

Would you like to (a) book a seat, (b) see the amount of available seats
or (c) update a flight? c
Enter the flight number: 3017
Enter the new status (o) on-time, (d) delayed or (c) cancelled: d
```

Air Flight Company Project - Step 3



```
Would you like to (a) book a seat, (b) see the amount of available seats
or (c) update a flight? a
Enter the flight number: 3527
Seat booked!
Would you like to (a) book a seat, (b) see the amount of available seats
or (c) update a flight? a
Enter the flight number: 3000
Flight not found.
Would you like to (a) book a seat, (b) see the amount of available seats
or (c) update a flight? a
Enter the flight number: 3017
Seat booked!
Would you like to (a) book a seat, (b) see the amount of available seats
or (c) update a flight? b
Available seats on flight 3527 to Madrid (on-time): 852
Available seats on flight 3017 to Paris (delayed): 219
```

Interface

An **interface** is a completely "abstract class" that is used to group related methods with empty bodies.

To access the interface methods, the interface must be "implemented" by another class with the **implements** keyword.

On implementation of an interface, you must **override all** of its methods.

Interface

- Can contain **constants** but not **properties**, interface attributes are by default public, static and final

- Can contain **abstract** methods, **default** methods and **static** methods

- An interface **cannot contain a constructor** (as it cannot be used to create objects)

Interface

```
public interface VehicleOperations {

  // Constant
  int MAX_SPEED = 120; // Maximum speed in km/h

  // Method signature (abstract method)
  void startEngine();

  // Method signature (abstract method)
  void stopEngine();

  // Default method
  default void turnOnLights() {
    System.out.println("Lights are turned on.");
  }
```

```
  // Default method
  default void turnOffLights() {
      System.out.println("Lights are turned off.");
  }

  // Static method
  static boolean isValidSpeed(int speed) {
    return speed >= 0 && speed <= MAX_SPEED;
  }
```

Interface

- It is a contract with a class. It specifies what needs to be implemented.

- Interfaces allow **multiple** inheritance

- Provides greater flexibility

Now YOUR TURN !

Let’s do exercise 1

Abstract

- Cannot be instantiated
- Can contain implemented methods
- Can contain abstract methods

Abstract

```java
public abstract class Shape {
  String color;

  // Constructor
  public Shape(String color) {
    this.color = color;
  }

  // Abstract method
  public abstract double getArea();

  // Concrete method
  public String getColor() {
    return color;
  }
}
```

```java
public class Circle extends Shape {
  private double radius;

  public Circle(String color, double radius) {
  super(color);
  this.radius = radius;
  }

  @Override
  public double getArea() {
    return Math.PI * radius * radius;
  }
}
```

Non-Access modifiers, for classes

- **final** – The class cannot be inherited by other classes.

- **abstract** – The class cannot be used to create objects (To access an abstract class, it must be inherited from another class.

Non-Access modifiers, for attributes, methods and constructors

- **final** – Attributes and methods cannot be overridden/modified.
- **static** – Attributes and methods belongs to the class, rather than an object.
- **abstract** – Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example abstract void run();. The body is provided by the subclass (inherited from).

Abstract/Static example

```java
public abstract class CompanyAnimal {

   private String name;
   private int age;
   private static int animalCount = 0;

   public CompanyAnimal() {
       name = "";
       age = 0;
       animalCount += 1;
   }

   public static int getAnimalCount() {
       return animalCount;
   }
}
```

PARENT

```java
public class Dog extends CompanyAnimal {
    public Dog() {
        super();
    }
}
```

CHILD

```java
public class Cat extends CompanyAnimal {
   public Cat() {
       super();
   }
}
```

CHILD

Abstract/Static example

```
public class MainProgram {
   public static void main(String[] args) {
       System.out.println("There are " + CompanyAnimal.getAnimalCount() + "
animals.");
       Dog doggo = new Dog();
       System.out.println("There are " + doggo.getAnimalCount() + "
animals.");
       Cat missy = new Cat();
       System.out.println("There are " + CompanyAnimal.getAnimalCount() + "
animals.");
       System.out.println("There are " + doggo.getAnimalCount() + "
animals.");
   }
}
```

output

```
There are 0 animals.
There are 1 animals.
There are 2 animals.
There are 2 animals.
```

Now YOUR TURN !

Let’s do exercise 2

Air Flight Company Project - Step 4

- Update the Aircraft class to be abstract.

- Create an interface called Bookable. It will specify that certain class/es can be booked. It should be possible to know how many current bookings there are and how many are left. The interface should also provide a method to make a new booking.

- Which existing class should be updated to implement the Bookable interface?

Air Flight Company Project - Step 4

- Add a new feature to the project. The new class should also implement the Bookable interface.
    - Maybe passengers can book first class seats?
-
    - Maybe passengers can book special types of foods?
-
    - Maybe each flight needs to book a gate at the airport?