Welcome to the

Java Course

Module 3 – Day 03

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 2

- Create an ENUM to represent the flight status, the statuses could be ON_TIME, DELAYED, CANCELLED

- Add a property to the “Flight” class to store the flight’s status.

- Whenever a new Flight gets created, its status should be ON_TIME

Air Flight Company Project - Step 2

- Modify the main program to allow adding multiple flights.
- The main program should also allow updating the status of a flight

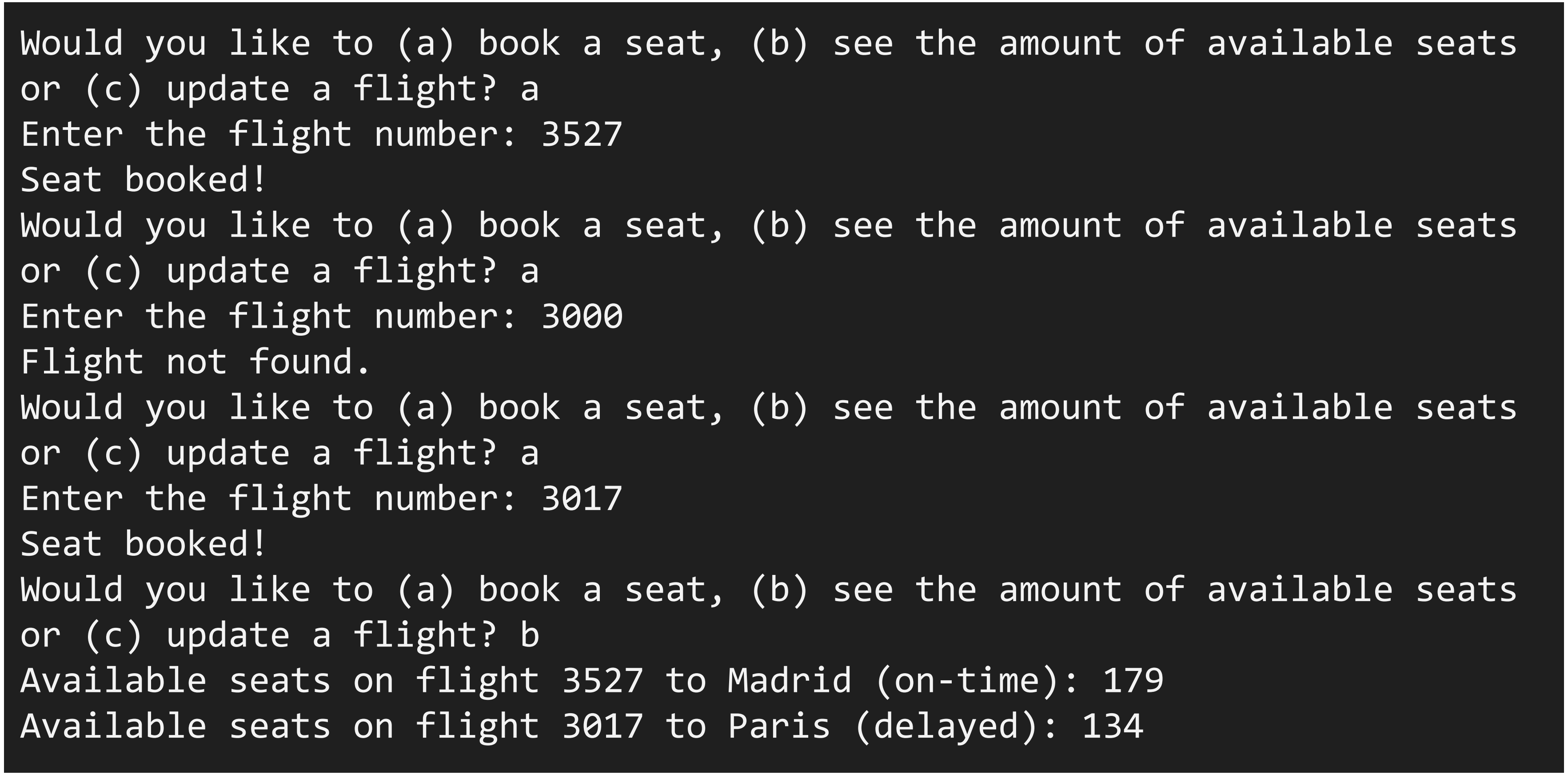
Air Flight Company Project - Step 2

```
>>> New Flight <<<
Enter flight number: 3527
Enter destination: Madrid
Enter flight capacity: 180
Flight created. Would you like to add another flight (y/n)? y

>>> New Flight <<<
Enter flight number: 3017
Enter destination: Paris
Enter flight capacity: 135
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 2



```
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): 179
Available seats on flight 3017 to Paris (delayed): 134
```

Encapsulation

- In Object-Oriented Programming, one of the fundamental concepts is **encapsulation**.
- It involves restricting and managing the **visibility** of the contents of an object.

Encapsulation

- We've already seen one aspect of encapsulation, which is setting attributes as **private**.
- This prevents interacting with them **outside** of the class.
- Enhances **data protection**.

Encapsulation

- With encapsulation, we interact with the class solely through access methods.
- This allows us to **modify** the class without affecting the rest of the code.
- This way, **maintenance**, **updates**, and **debugging** are more straightforward and more flexible.

Encapsulation

```
public class BankAccount {
  private Person owner;
  private String accountNumber;
  private BIC bic;
  private double balance;
  private String pinCode;

  public BankAccount(Person owner, String accountNumber,
BIC bic) {
    this.owner = owner;
    this.accountNumber = accountNumber;
    this.bic = bic;
  }

  private boolean connect() {
    int pin = Scanner.nextLine();
    if (pin.equals(pinCode))
      return true;
    return false;
  }
```

```
public void deposit(double amount) {
  if (amount > 0)
    balance += amount;
  }

public boolean withdraw(double amount) {
  if (connect() && balance - amount >= 0)
    return true;
  else
    return false;
}
```

```
public class AmazonAccount extends BankAccount {
  private Person owner;

  public void isSameOwner(){
    return owner.equals(super.owner);
  }


}
```

Access modifiers, for classes

- **public** – The class is accessible by any other class
- **default** – The class is only accessible by classes in the same package. This is used when you don't specify a modifier.

Access modifiers, for attributes, methods and constructors

- **public** – The code is accessible for all classes.
- **private** – The code is only accessible within the declared class.
- **protected** – The code is accessible in the same package and subclasses.
- **default** – The code is only accessible in the same package. This is used when you don't specify a modifier.

Accessibility

```
public class CompanyAnimal {
  private String name;
  private int age;

  public CompanyAnimal(){
    this.name = “”;
    this.age = “”;
  }

 public getName(){
   return name;
  }

 public setName(String name){
  this.name = name;
  }

}
```

PARENT

```
public class Dog extends CompanyAnimal {
  private DogBreads breed;

  public Dog(){
    super();
    this.breed = DogBreads.CHIHUAHUA;
  }

  @Override
  public String toString() {
    return ”Dog name is " + super.getName();
  }
}
```

CHILD

```
public static void main (String[] args){
  private Dog doggo = new Dog();

 doggo.setName(“Doggo”);
 System.out.println(doggo.toString());

}
```

Accessibility

```
public class CompanyAnimal {
  public String name;
  public int age;

  public CompanyAnimal(){
    this.name = “”;
    this.age = “”;
  }
}
```

PARENT

```
public class Dog extends CompanyAnimal {
  private DogBreads breed;

  public Dog(){
    super.name = “”;
    super.age = 0”;
    this.breed = DogBreads.CHIHUAHUA;
  }
}
```

CHILD

Now **YOUR TURN !**

Let’s do exercises 1 and 2

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
```