Fundamentals of Object Oriented Programming in Java

WHAT IS OBJECT-ORIENTATION?

Maurice Naftalin
Java Champion, JavaOne Rock Star
Author: *Mastering Lambdas, Java Generics and Collections*
@mauricenaftalin

What is Object Orientation?

Based on *objects*, which consist of

- State (data, held in fields)
- Behavior (methods, functions)

The state is hidden from other objects

- data is accessed through the object's own methods

An OO program is a collection of objects working together

Why Object Orientation?

Almost universal in the software industry

Useful for analysis, design, and coding

Most successful for managing complexity

- Supports decoupling (reduced interdependence) of subsystems
- Encourages reuse of components

Design rationale for much of Java language

What Do You Need to Know?

This is a Java course

Be comfortable with Java:

- Syntax
- Collections framework
- Java 8 streams

Some OO knowledge an advantage

Course Summary

Overview of object oriented design

- Introduction to the course application

Encapsulation and abstraction

- Establishing boundaries within a system

Inheritance and polymorphism

- Making a system extensible

Interfaces and system design

- The principles of good OO programming

Module Overview

The Starting Point: Use Cases

From Use Cases to Conceptual Classes

Aggregation: the Has-A Relation

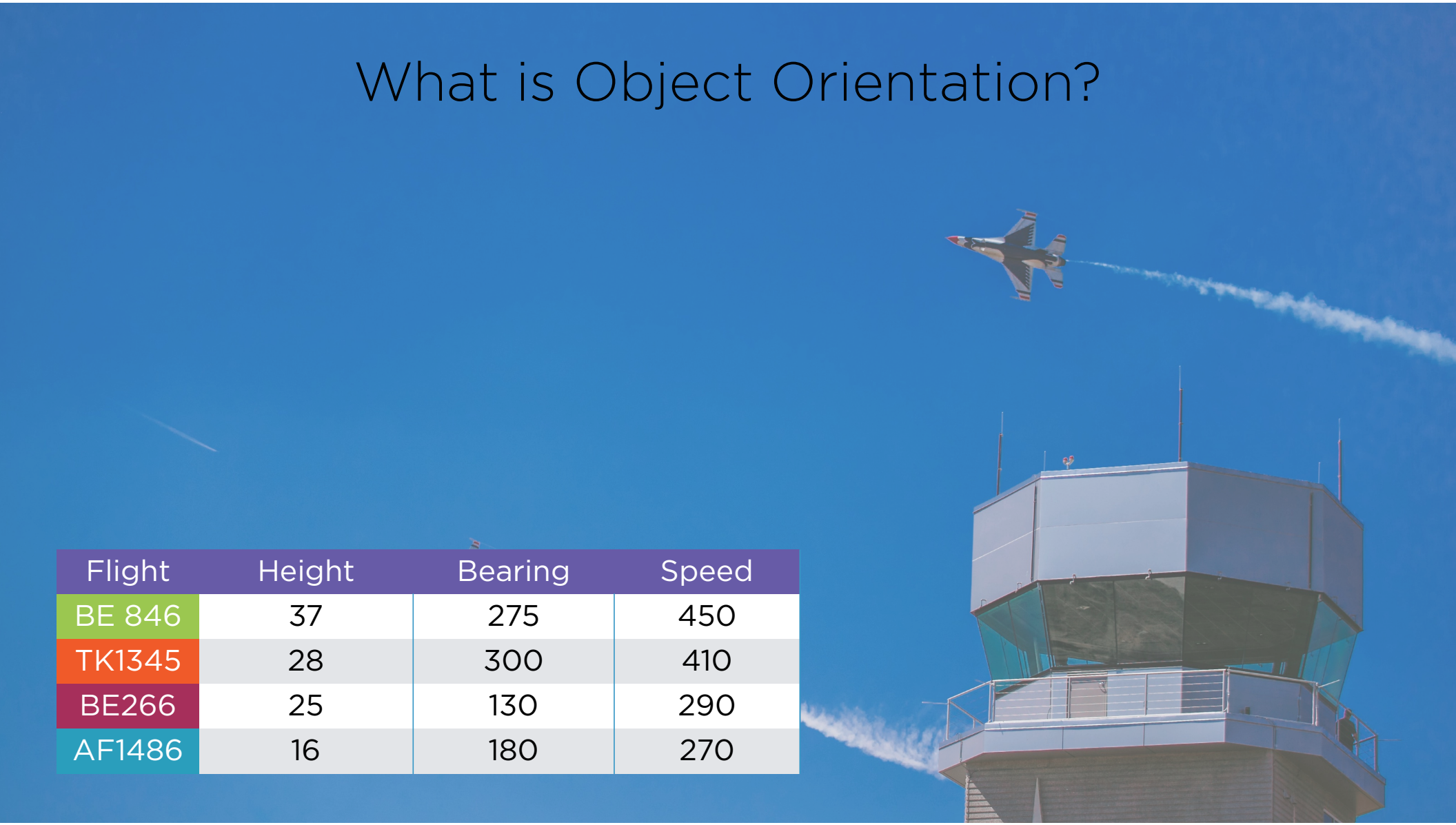
Inheritance: the Is-A Relation

Demo: Code from Conceptual Classes

What about Static Members?

What Is Object Orientation?

What is Object Orientation?



Flight	Height	Bearing	Speed
BE 846	37	275	450
TK1345	28	300	410
BE266	25	130	290
AF1486	16	180	270

What is Object Orientation?

What's Wrong with Global Data Structures

Responsibility for state becomes spread out

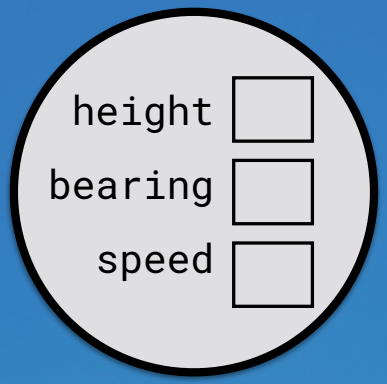
Different subsystems are coupled together

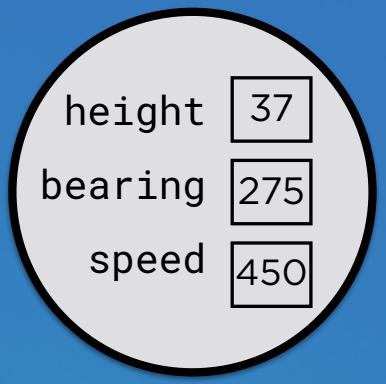
Difficulty representing varied data formats

Extremely difficult to parallelize

Flight	Height	Bearing	Speed
BE 846	37	275	450
TK1345	28	300	410
BE266	25	130	290
AF1486	16	180	270

What is Object Orientation?

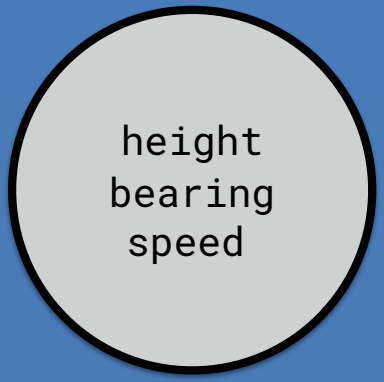




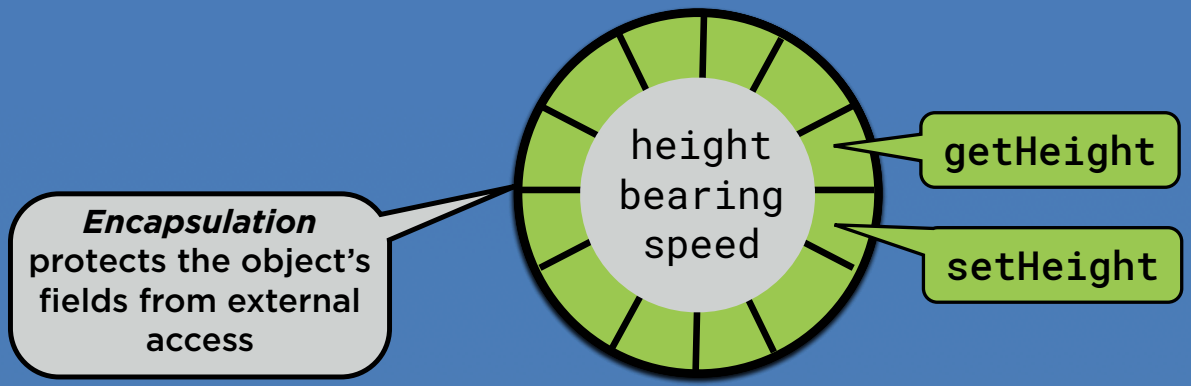
BE 846

Flight	Height	Bearing	Speed
BE 846	37	275	450
TK1345	**28**	300	**410**
BE266	25	130	290
AF1486	16	180	270

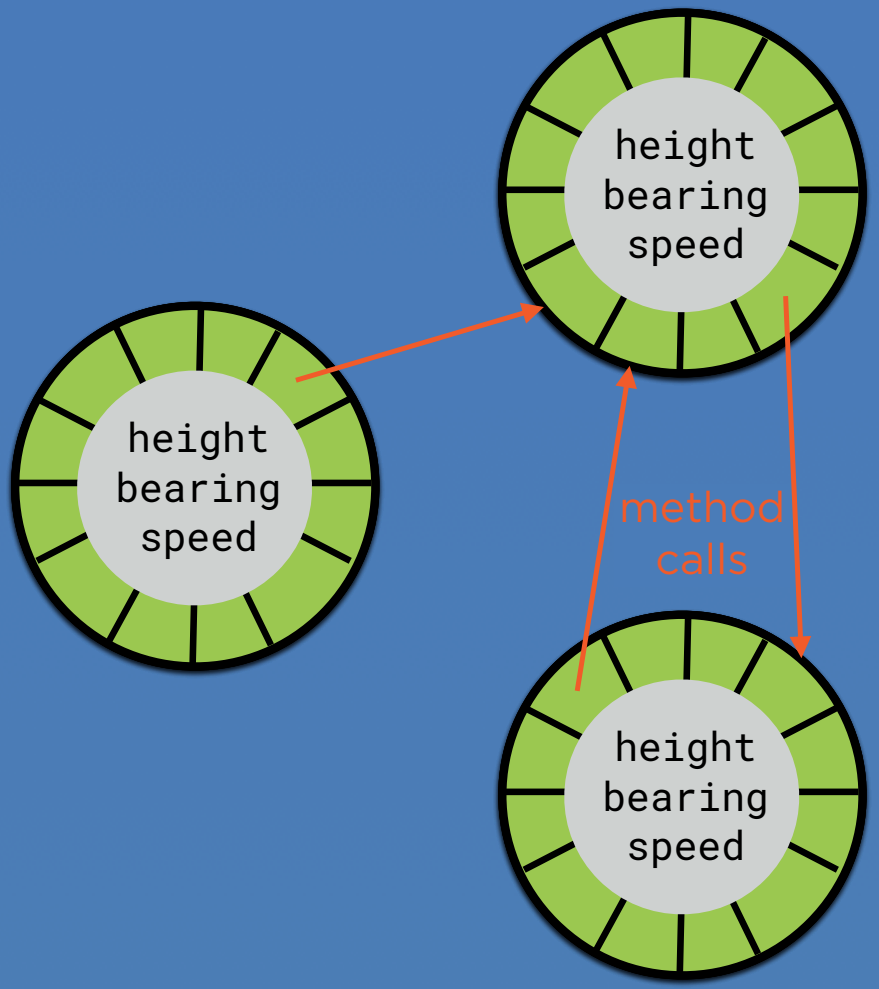
What is Object Orientation?



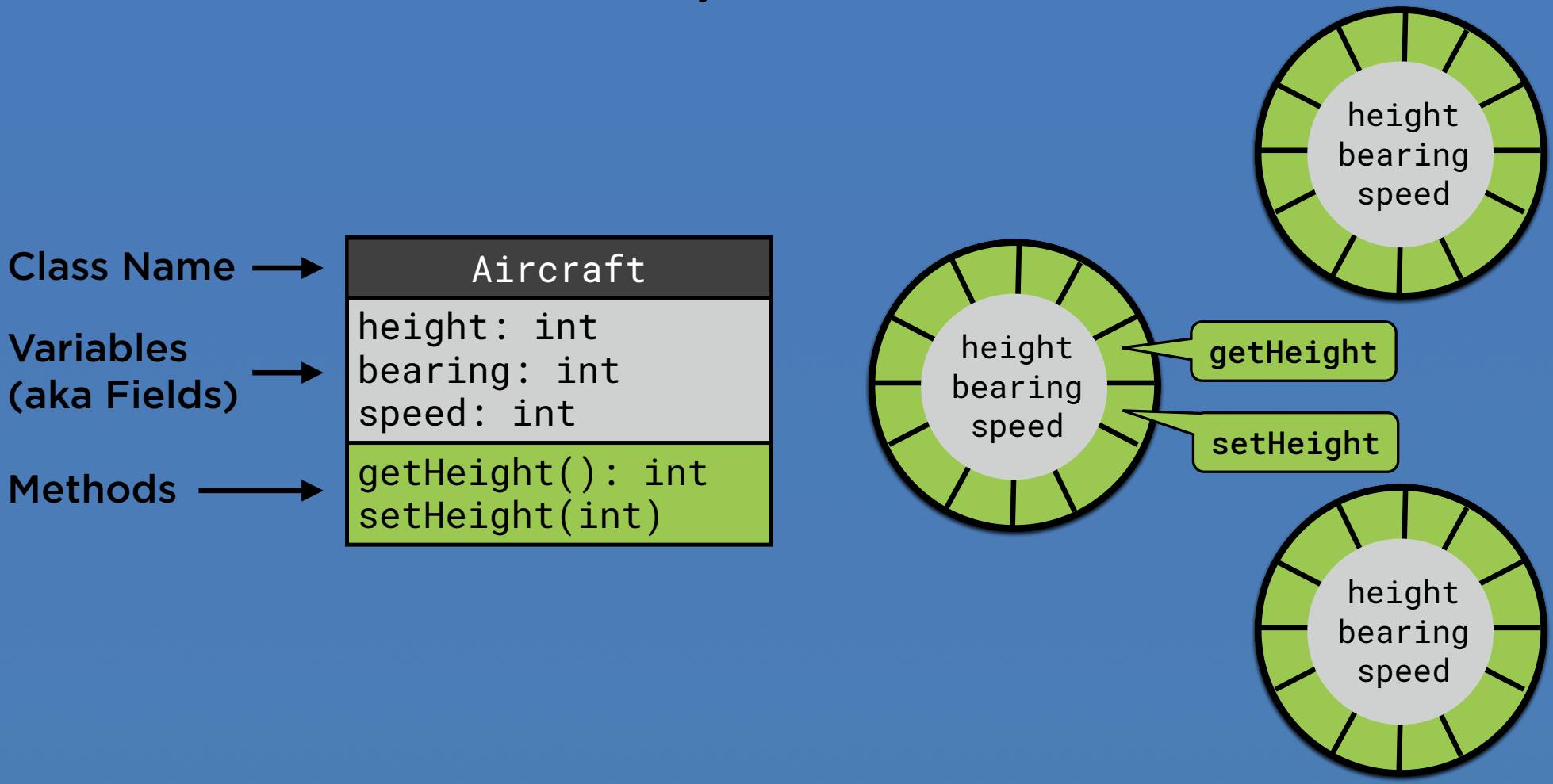
What is Object Orientation?



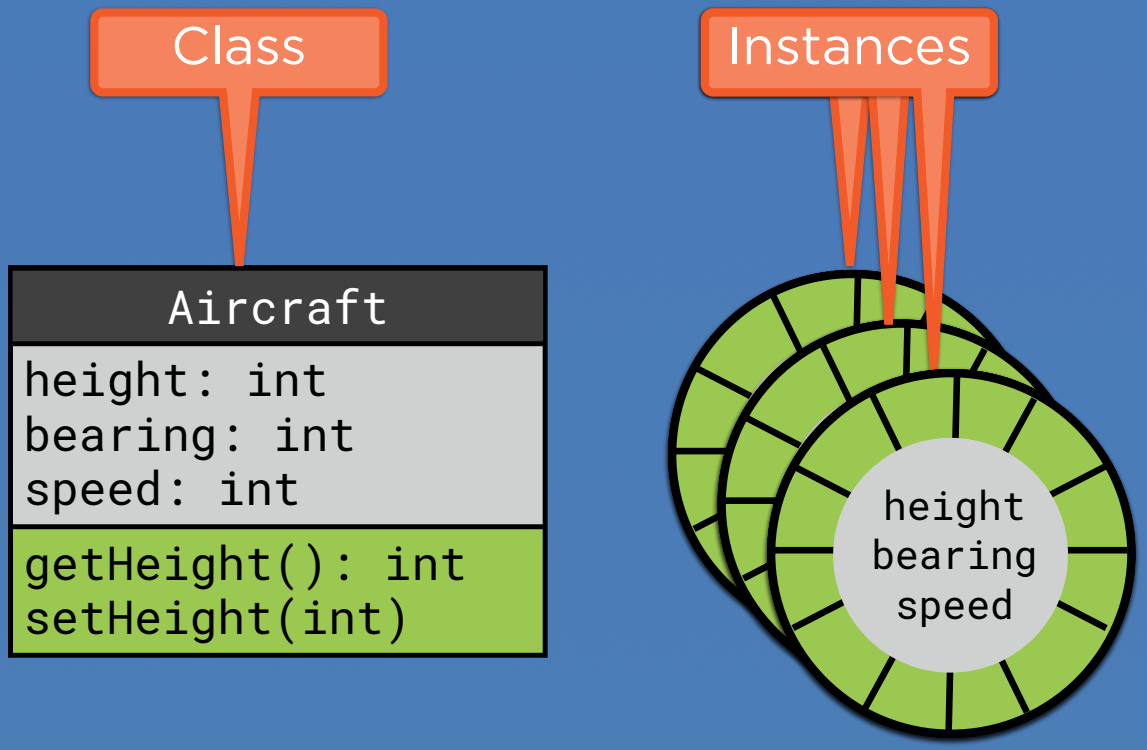
What is Object Orientation?



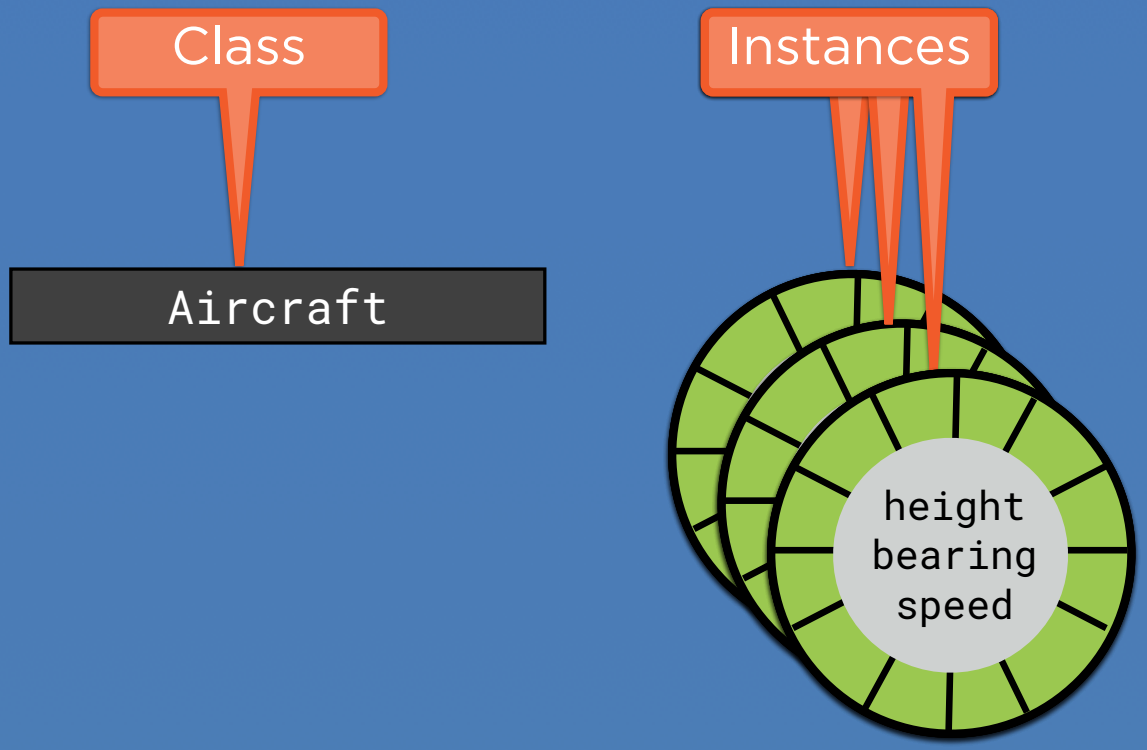
What is Object Orientation?



What is Object Orientation?



What is Object Orientation?



Use Cases

Scenarios of system operation

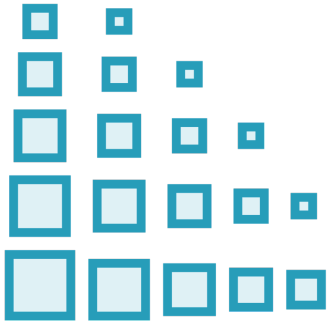
Often started by a human user

List the sequence of interaction steps

1. Create Order and Check Out

"A customer creates an order by first adding various products (digital or physical) to a shopping cart, then checks out, making a payment using a credit card."

2. Fulfill order



“For an order, the system discovers which distribution centers hold stock of the products in the order. Each centre which can help fulfill the order is sent details of the products required together with the customer’s details.”

Use Cases

Domain Model

which consists of

Conceptual Classes

Software Classes

Discovering Conceptual Classes

"A *customer* creates an *order* by first adding various *products* (digital or physical) to a *shopping cart*, then checks out, making a *payment* using a *credit card*."

Customer

Order

Product

Shopping Cart

Payment

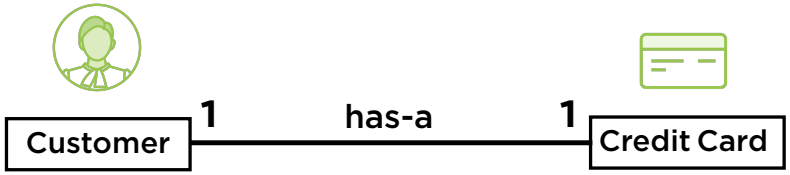
Credit Card

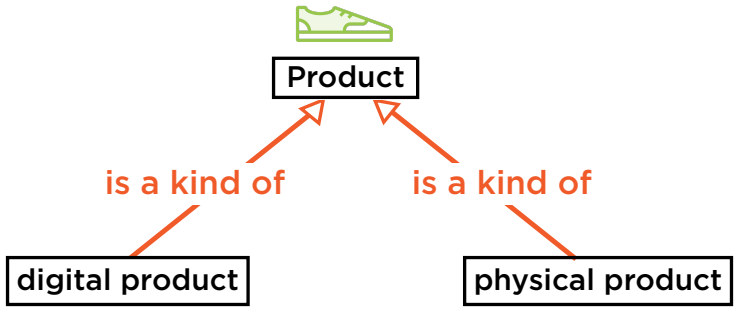
Class Relations

Class Relations

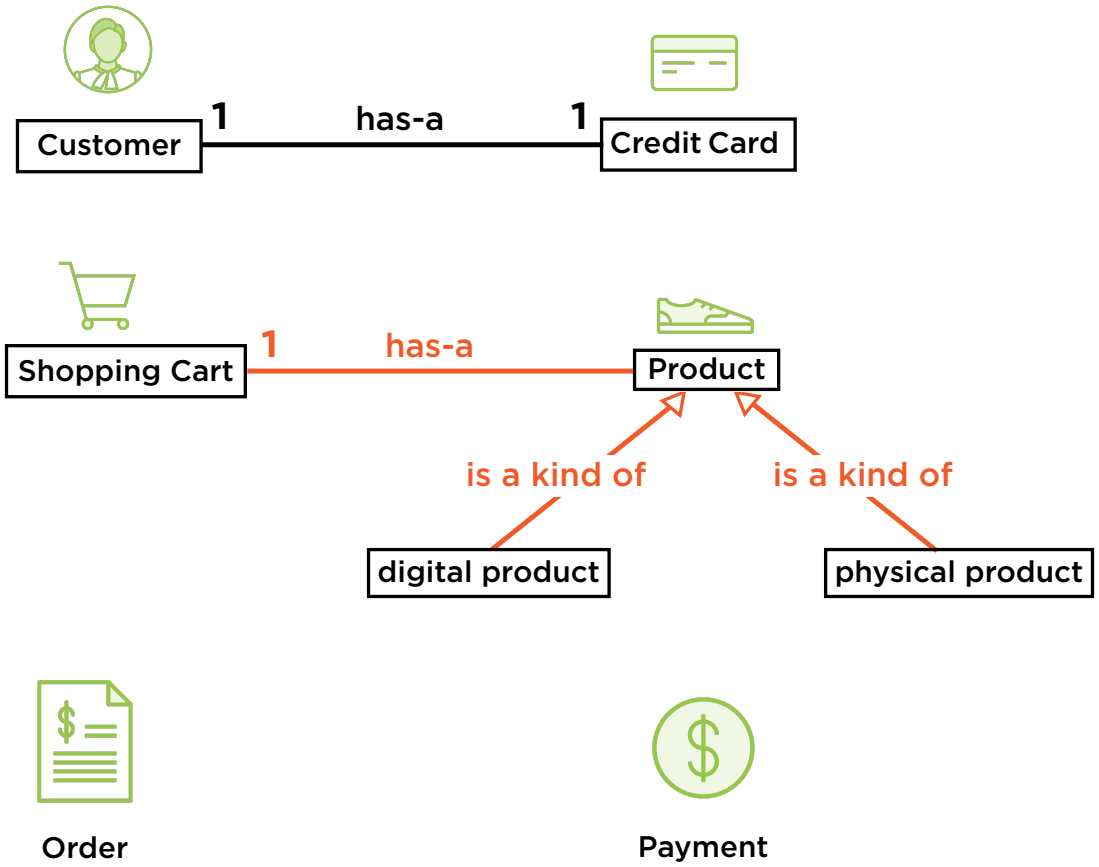
Class Relations: **“has-a”**

Class Relations: **“is-a”**

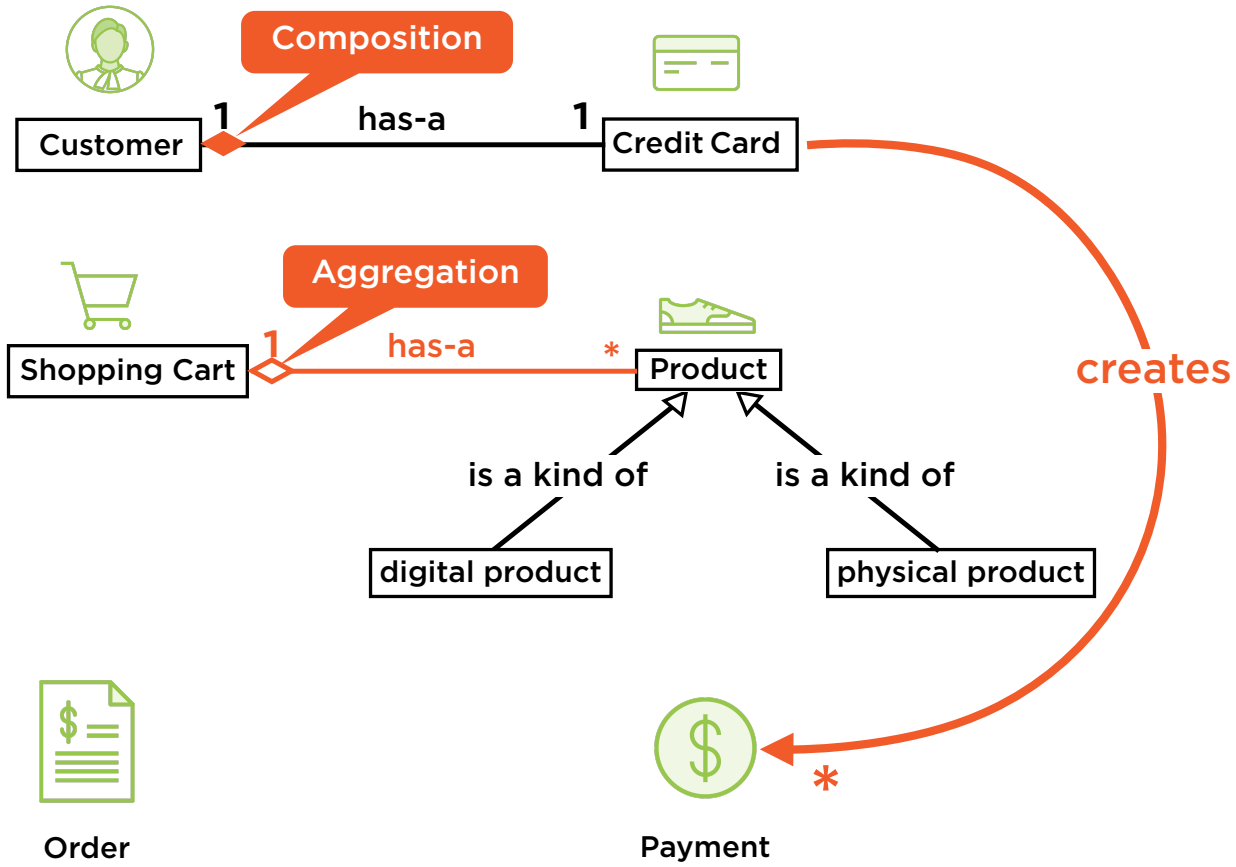




Class Relations:

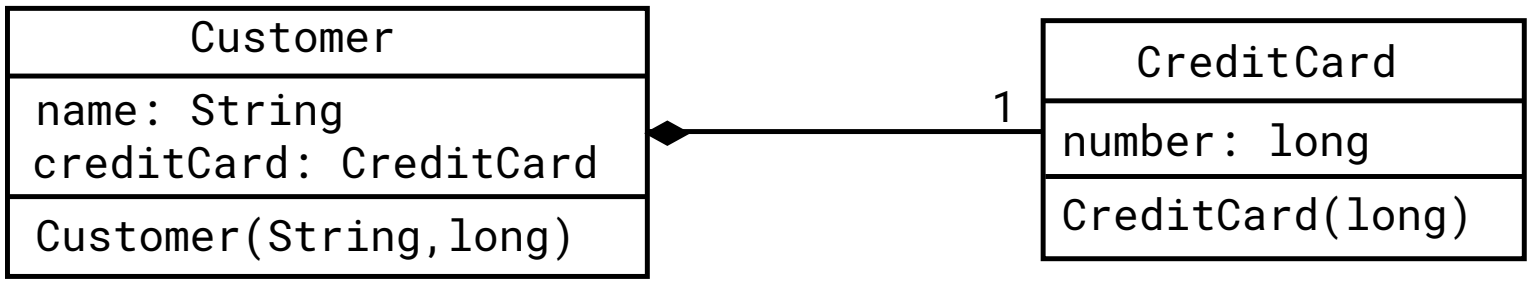


Class Relations



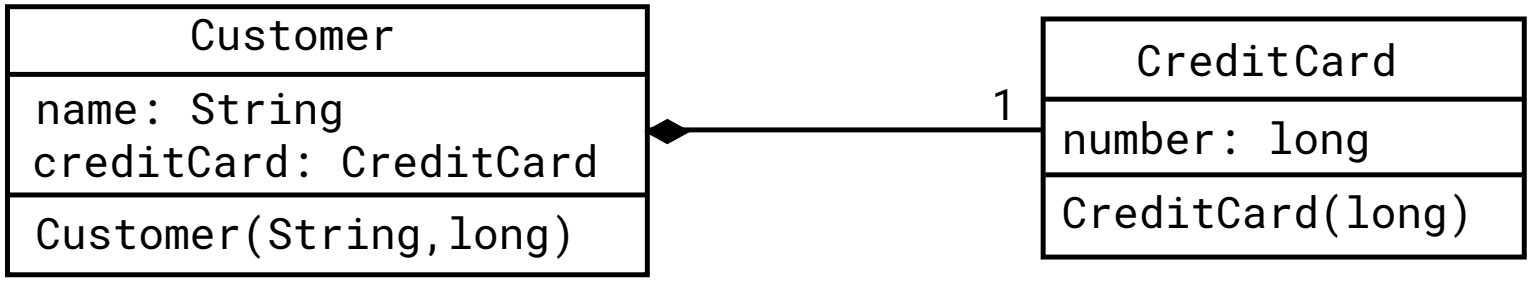
Software Classes

"A customer ... checks out, making a payment using a credit card."

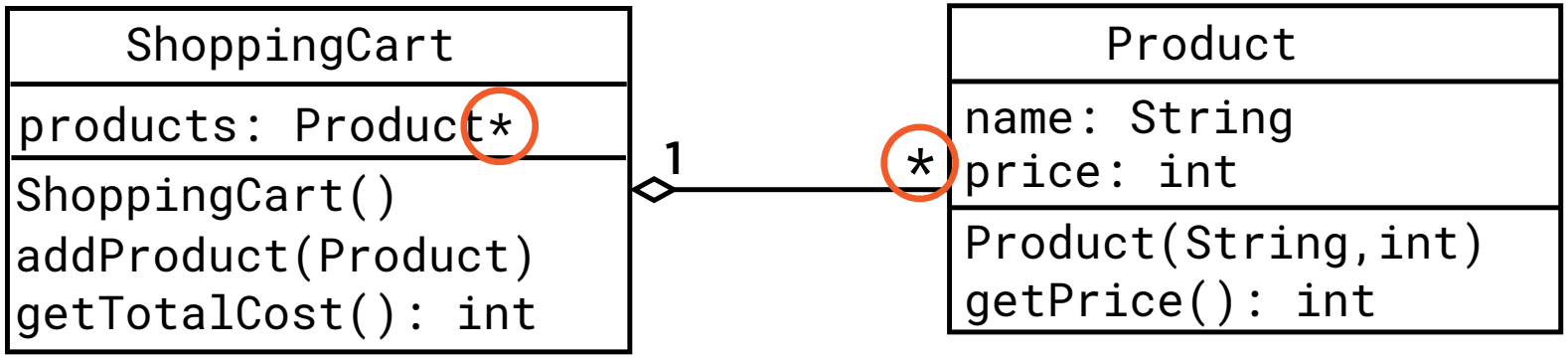


```java
public class Customer {

    private final String name;
    private CreditCard creditCard;

    public Customer(String name,
                    long ccNumber) {
        this.name = name;
        this.creditCard =
            new CreditCard(ccNumber);
    }
}
```

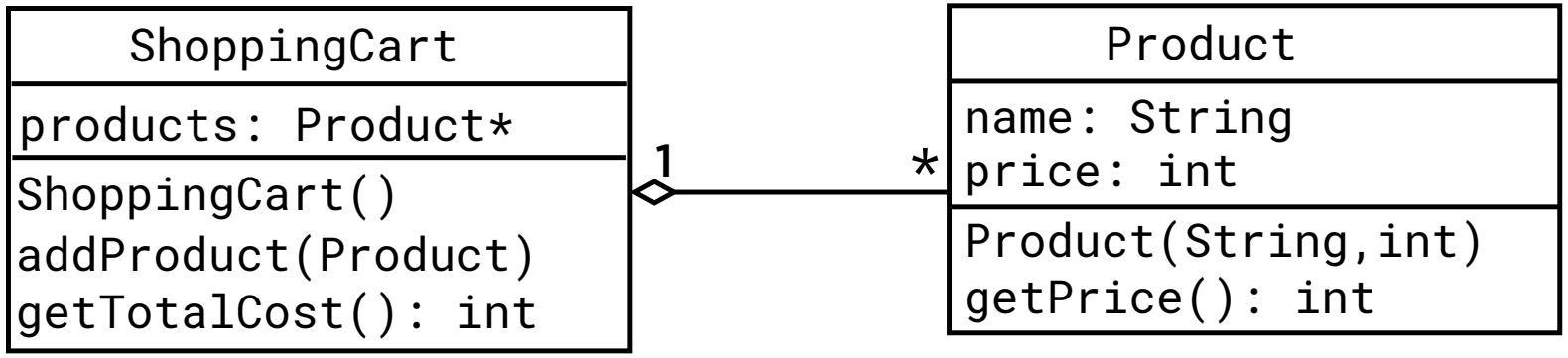
```java
class CreditCard {

    private final long cardNumber;

    CreditCard(long cardNumber) {
        this.cardNumber = cardNumber;
    }
}
```



Software Classes

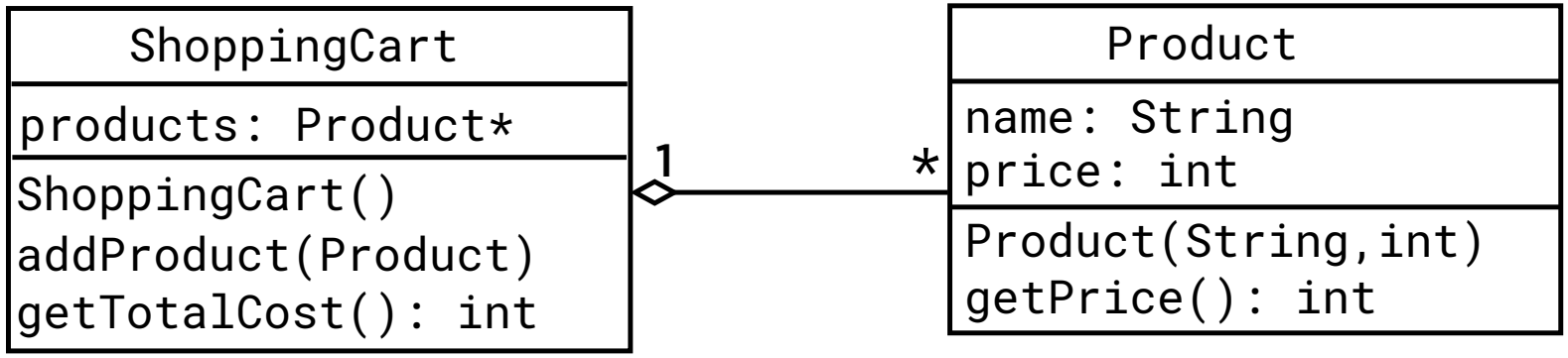


```java
public class Product {

    private final String name;
    private int price;

    public Product(String name,
                   int price) {
        this.name = name;
        this.price = price;
    }

    public int getPrice() {
        return price;
    }
}
```



```java
public class ShoppingCart {

    private List<Product> products = new ArrayList<>();

    public void addProduct(Product product) {
        products.add(product);
    }

    public int getTotalCost() {
        return products.stream()
            .mapToInt(Product::getPrice)
            .sum();
    }
}
```

```java
public class Product {

    private final String name;
    private int price;

    public Product(String name,
                   int price) {
        this.name = name;
        this.price = price;
    }

    public int getPrice() {
        return price;
    }
}
```



Demo: Exercising the Classes

Let's use the classes we've created

They don't have much functionality yet

- But we can construct instances
- And use the limited function they do have

Finally, we'll see how static code fits in

Module Summary

- Why Object Orientation?
- The Starting Point: Use Cases
- From Use Cases to Conceptual Classes
- Aggregation: the Has-A Relation
- Inheritance: the Is-A Relation
- Demo: Code from Conceptual Classes
- What about Static Members?

Up Next:

Encapsulation and Abstraction