Polymorphism

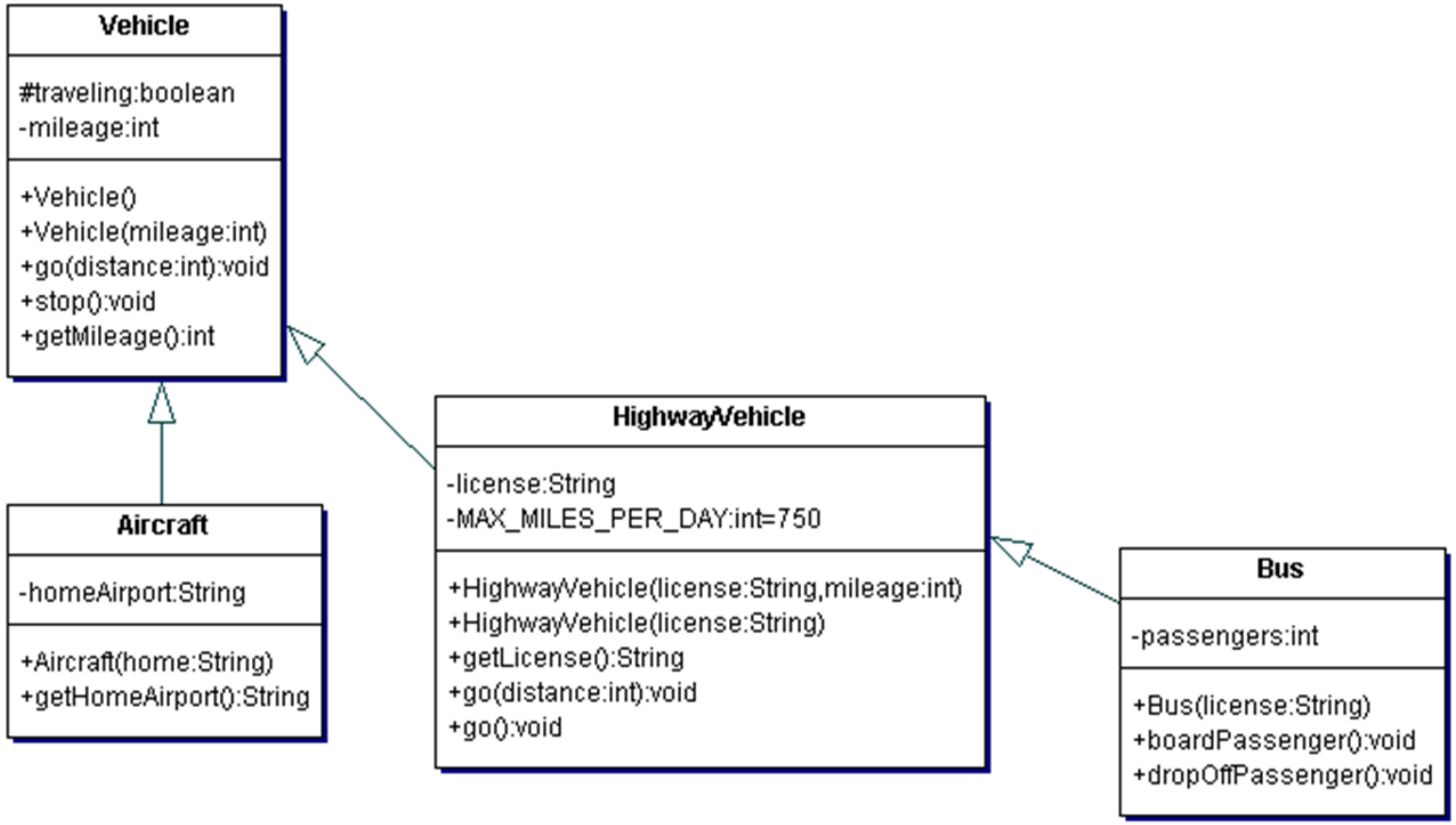
Definition and concepts, substitution principle, casting, formal parameters, interfaces vs. inheritance

Introduction

Polymorphism:

- Means creating an object that has more than one form. Two root words: poly = many, morph = form.
- Occurs in the context of interfaces and inheritance.
- Principle: An object behaves according to what type it actually *is*, not according to how it was declared.
- Also called **runtime binding** or **late binding** since you cannot always tell the type of an object until runtime.

Reviewing the class hierarchy



Substitution principle

Whenever a base class object is expected, a child class type object can be substituted.

Examples:

```
public void methodFoo(Vehicle x) {
//…. }

//……

Vehicle car;
car = new HighwayVehicle(“123 NC”);
methodFoo(car);

HighwayVehicle truck = new HighwayVehicle(“456 HI”);
methodFoo(truck);
```

Illegal assignments/method calls

Assume this:

```
public void methodBar(HighwayVehicle x) {
//…. }
```

Here is what you *cannot* do:

```
Vehicle car = new Vehicle();
methodFoo(car)  // ILLEGAL

HighwayVehicle truck;
truck = new Vehicle( ); // ILLEGAL

Vehicle jet = new Airplane(“RDU”);
String s = jet.getHomeAirport();  // ILLEGAL
```

Casting

Polymorphism: an object behaves according to its actual type, not its declared type.

But this code is illegal because the compiler assumes the type of jet is Vehicle:

```
Vehicle jet = new Airplane(“RDU”);
String s = jet.getHomeAirport();  // ILLEGAL
```

You can tell the compiler the actual type of an object by **casting**.

```
String s = ((Airplane)jet). getHomeAirport(); // Legal
```

Casting syntax

```
Vehicle jet = new Airplane(“RDU”);
String s = ((Airplane) jet).getHomeAirport();
```

cast member access

The member access operator (the dot) has higher precedence than the cast, which is why the outer parentheses. This is also called a downcast, since you are casting down the class hierarchy.

You can also cast up, but there is never a need. That’s because of the substitution principle.

Casting exceptions

With a cast, you tell the complier the actual type of the object. But you have to be honest. Look at this code:

```
Vehicle jet = new Vehicle();
if (<condition that could vary>)
      jet = new Airplane(“RDU”);
String s = ((Airplane) jet).getHomeAirport();
```

The last statement will compile, and it runs without error when the condition is true. But when the condition is false, java throws a ClassCastException at runtime.

Defining go( ) in the Vehicle hierarchy

Vehicle

```
public void go(int distance) {
   traveling = true;
   mileage += distance;
}
```

HighwayVehicle

```
public void go(int distance) {
  if (distance < MAX_MILES_PER_DAY)
     super.go(distance);
 }

public void go() {
   go(100);
}
```

Bus

```
public void go(int x)
{
   super.go(x);
   passengers = 0;
 }
```

Polymorphism and formal parameters

```
public void goSomewhere(Vehicle v, int howFar)
{
    v.go(howFar);
}
```

By declaring the formal parameter type for goSomewhere() to be Vehicle, you can call the method with any subclass of Vehicle. That builds some generality into the code.

Whenever you want a method to act on any class in an entire hierarchy, make the formal parameter type the same as the base for the hierarchy.

Polymorphism and formal parameters

```
public void goSomewhere(Vehicle v, int howFar) {
    v.go(howFar);
}

// …
   Vehicle myVehicle = new Vehicle();
   HighwayVehicle myCar = new HighwayVehicle("abcd");
   Bus yellowBus = new Bus("NC-SCHOOL");

   double randomNumber = Math.random();
   if (randomNumber >= .75)
       this.goSomewhere(yellowBus, 20);
   else if (randomNumber >= .25)
       this.goSomewhere(myCar, 850);
   else
       this.goSomewhere(myVehicle, 100);
```

Interfaces vs inheritance: Example interface and implementing classes

```
public interface ForSale {
    boolean isExpensive();
    double getCost();
    void setCost(double cost);
}

public class AirplaneTicket implements ForSale {
// …
public class PottedPlane implements ForSale {
// …
```

… and methods with interface type formal parameters

```
public void markdownPrice(ForSale x) {
   if (x.isExpensive())
      x.setCost(x.getCost() * 9/10);
   else
      x.setCost(x.getCost()* 3/4);
}
// …

ForSale tickets = new AirplaneTicket("RDU", "LHR", 1000);
ForSale item = new PottedPlant("petunia", 3);
this.markdownPrice(tickets);
this.markdownPrice(item);
System.out.println("New ticket price: $″ + tickets.getCost()
         + "\nNew Petunia price: $" + item.getCost());
```

Polymorphism works the same with interfaces as with inheritance … but it is often a little nicer.

Inheritance can be “brittle.” Changes in the base class propagate through the whole class hierarchy. Interfaces are not brittle. Classes that implement interfaces are in general independent.

If you want identical default behaviors for several classes, then inheritance may be the best choice. If you have a design that is likely to change or expand, interfaces are likely better.