Java Week 3 (1)

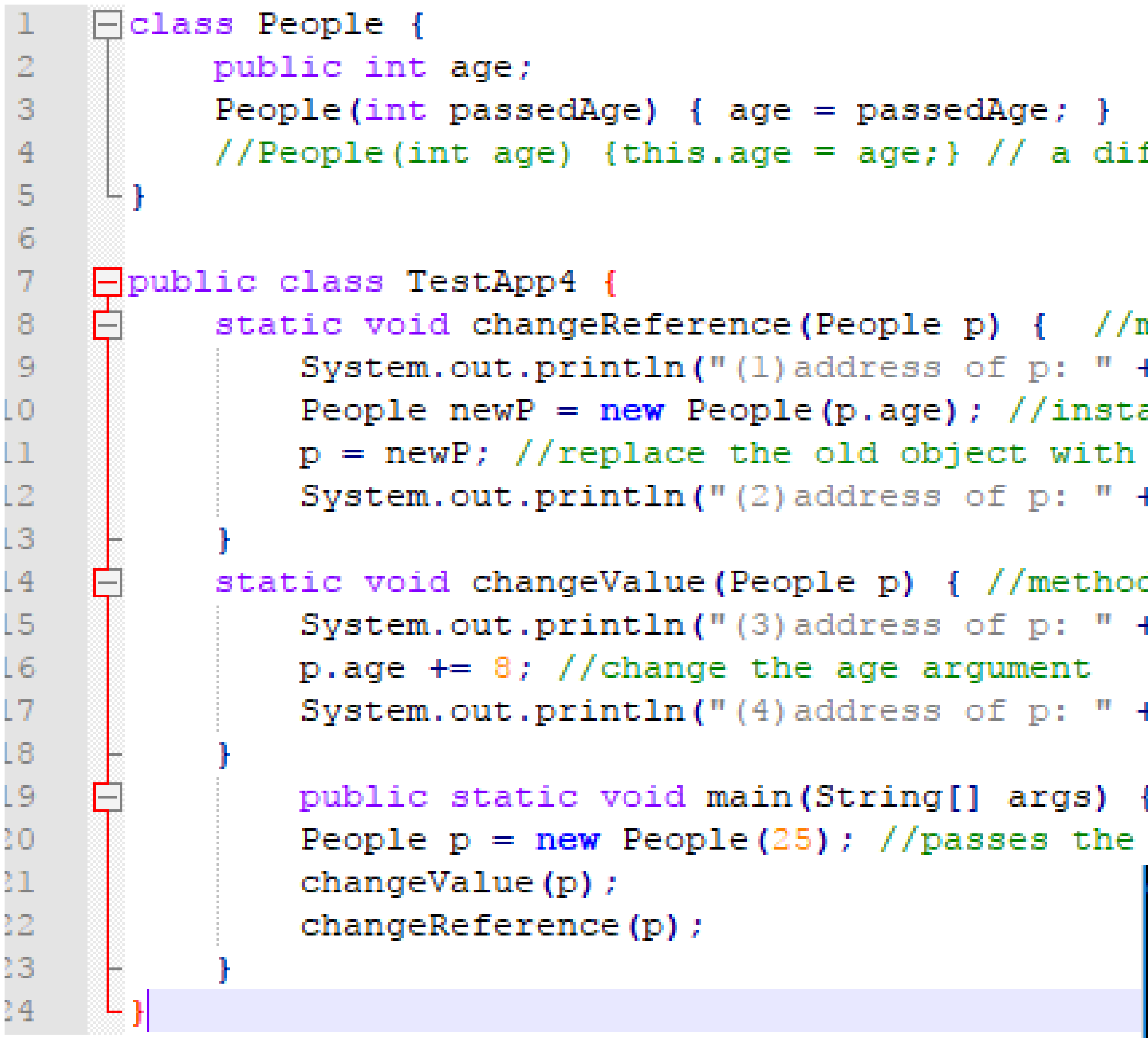
Review of key language constructs, introduction to OO and Inheritance

Dr Chris Roadknight

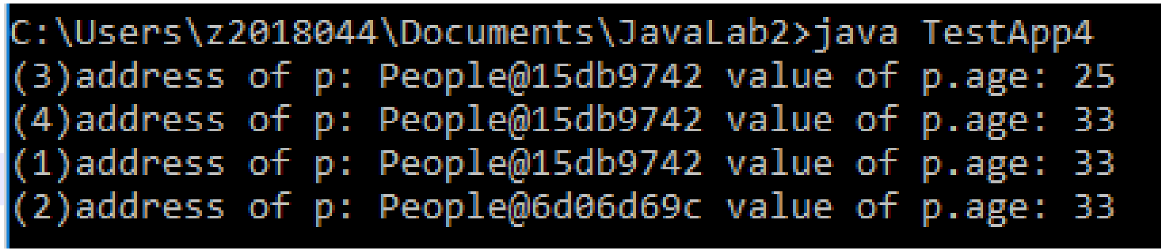
Pass by reference and pass by value

- The following code changes the location of a stored value
  - changeReference
- And changes the value itself without changing the locations
  - changeValue

- Note. We cheat a bit to display a memory ‘location’. If we print an object with no suffix it prints the originating class name and a Hash of some location



```java
class People {
    public int age;
    People(int passedAge) { age = passedAge; }
    //People(int age) {this.age = age;} // a different way to assign
}

public class TestApp4 {
    static void changeReference(People p) {   //method
        System.out.println("(1)address of p: " + p + " value of p.age: " + p.age);
        People newP = new People(p.age); //instance of a new object in new location
        p = newP; //replace the old object with the new one
        System.out.println("(2)address of p: " + p +" value of p.age: " + p.age);
    }
    static void changeValue(People p) { //method
        System.out.println("(3)address of p: " + p + " value of p.age: " + p.age);
        p.age += 8; //change the age argument
        System.out.println("(4)address of p: " + p + " value of p.age: " + p.age);
    }
        public static void main(String[] args) {
        People p = new People(25); //passes the value 25
        changeValue(p);
        changeReference(p);
    }
}
```



```
C:\Users\z2018044\Documents\JavaLab2>java TestApp4
(3)address of p: People@15db9742 value of p.age: 25
(4)address of p: People@15db9742 value of p.age: 33
(1)address of p: People@15db9742 value of p.age: 33
(2)address of p: People@6d06d69c value of p.age: 33
```

Constructors vs methods

- Constructors **initialise** objects when they are created.
- Methods **perform operations** on objects that already exist.
- Constructors are called **implicitly** when the **`new`** keyword creates an object.
- Methods can be called **directly** on an existing object.
- Constructors must be named with the **same** name as the class name.
- Methods must be called something **different** to the class name
- Constructors can't return anything (because the object itself is the thing being returned).
- Methods must be configured return something, although it can be void

Method overloading vs constructor overloading

```
class Overload {
        void display() {                    //these are methods
                System.out.println("you passed nothing");
        }
        void display(int x) {
                System.out.println("you passed " +x);
        }
}

class Overload{
        int data;                //these are constructors
        Overload(int x) { //object is created with one passed value
                data = x;
        }
        Overload(int x, int y) {
                data = x+y; //object is created with 2 passed values
        }
}
```

Static keyword summary

- Used to reduce JVM memory usage (very important in Java)
  - (aside.. JVM, memory, virtual memory, OS etc)
- Makes code safer (variables change in a specified way
- If a variable is declared `static` then changing it in any object changes it for all objects
  - Saves a lot of memory in some situation. (eg. agent based systems)
- Static methods are methods which don’t require an object of its class to be created before it can be called. Belongs to the class
- Static methods can’t use the `this` keyword as there is no instance for `this` to refer to.

Object Oriented Programming

- Aids decision making when designing an application
- 4 major concepts
- **Inheritance, Encapsulation, Abstraction and Polymorphism**
- Why use it:
  - Allows multiple developers to work on the same project more efficiently
  - Modular, Extensible, Reusable
- Why not use it:
  - Creating efficient Classes can be hard (hence we use `Dog` and `Car` so much)
  - Unforeseen interactions (emergent behavior!)
  - Performance issues and requires more code

Inheritance

- Allows for hierarchical program structures
- Means we can make a general class with common traits
- New classes can inherit these traits but also add class specific traits
- Class that is **inherited** is the superclass
- Class that **inherits** if the subclass
- Subclass inherits all the variables and methods of the superclass
- Changes in superclass effect subclasses
- Subclasses code can change the ‘intention’ of the superclass code
  - Immutable parameters classes can be copied and changed*

Inheritance

- A class can inherit all the methods and variables of another class (a "superclass").
- Use the keyword `extends`

```
class Superclass {…..}
class Subclass extends Superclass { .....}
```

- The subclass extends the superclass by adding behavior and data to the behavior and data provided by the superclass

The `Object` class is the root superclass

- Every class is a subclass of Java's `Object` class..unless overridden
- Therefore every class includes the following methods:
  - toString()
  - Equals(Object x)
  - Hashcode()
  - getClass()
- Every class (other than object) is a subclass
- Note-So If p is an object, java automatically converts "p= " +p

To "p= " + p.toString()

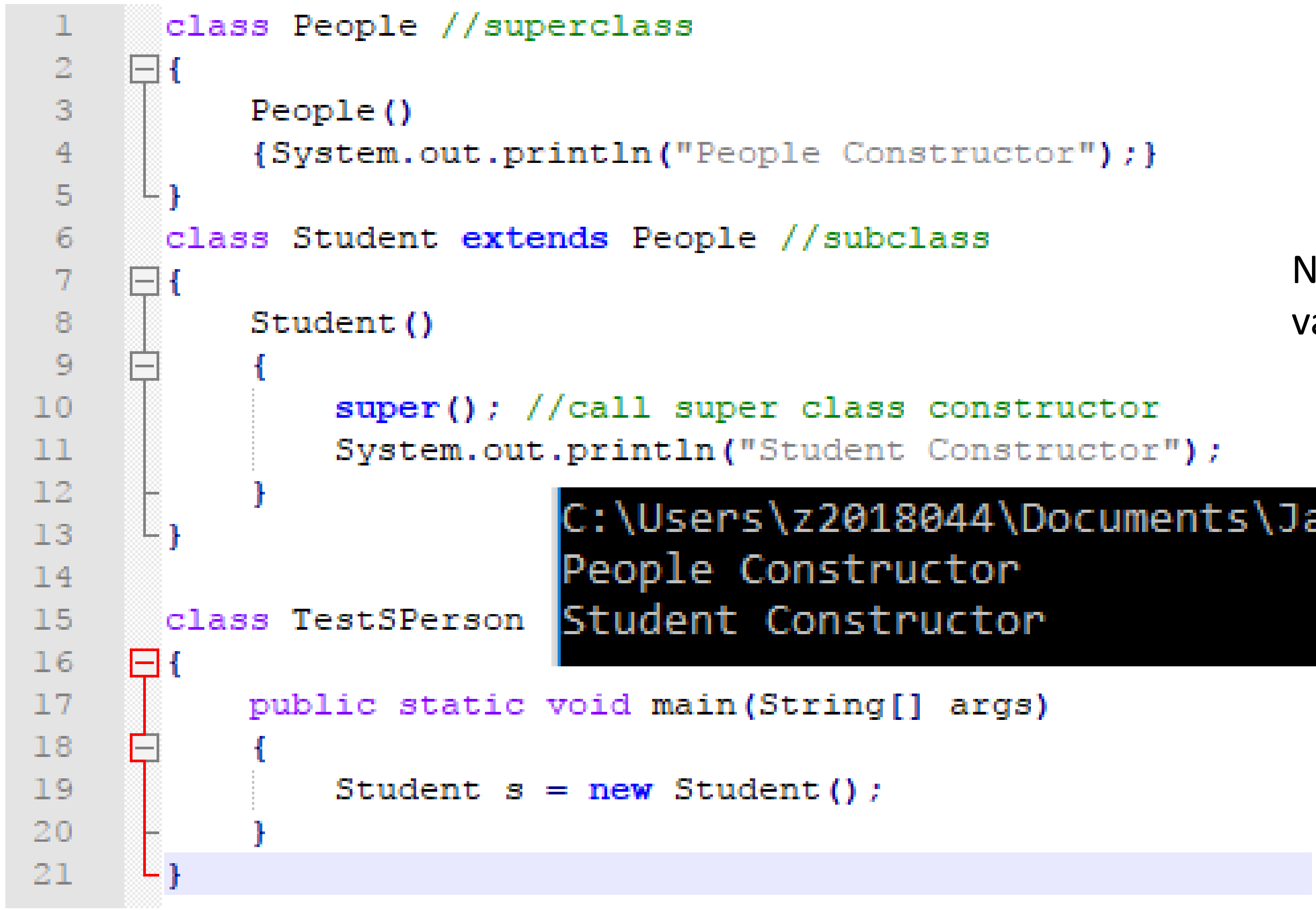
Example

```
class OneDimPoint { //Superclass
        int x = 3;
        int getX() { return x; }
        }
class TwoDimPoint extends OneDimPoint { //Subclass
        int y = 4;
        int getY() { return y; }
        }
class TestInherit {
        public static void main(String[] args) {
                TwoDimPoint pt = new twoDimPoint();
                /*this object has x and y, even though the
                subclass has no getX method*/
                System.out.println(pt.getX() + "," + pt.getY());
        }
}
```

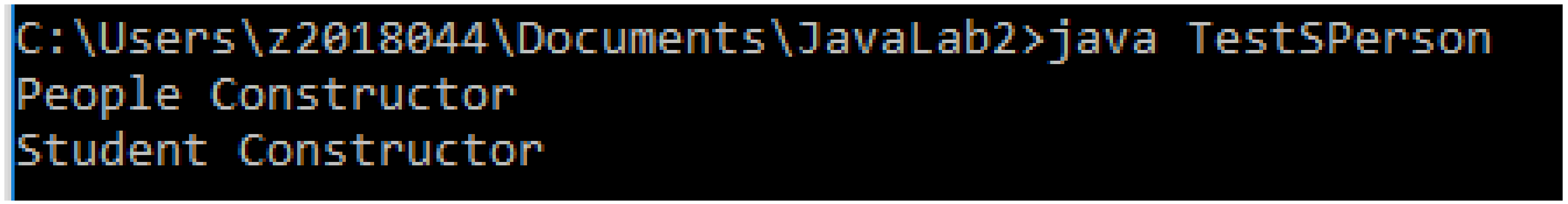
Inheritance key points

- Each superclass can have many subclasses but in Java (as opposed to C++) each class can have **at most** one superclass.
- A subclass cannot access the private members of its superclass.
- A subclass constructor can call a superclass constructor by use of super( ), before doing anything else.
- If you do not call a superclass constructor, the no-argument constructor is automatically called.

Superclass constructors ((over)simple example)



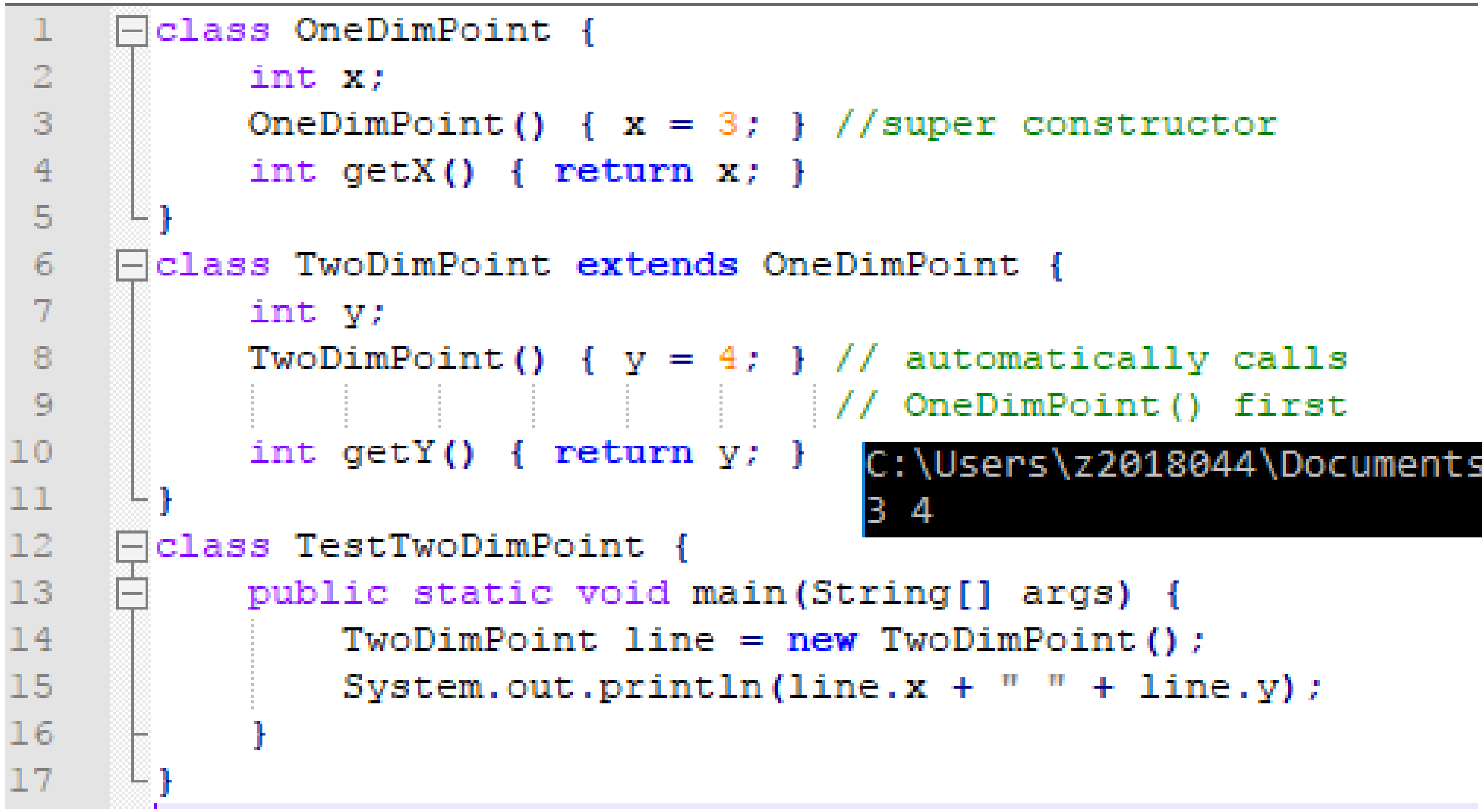
```java
class People //superclass
{
    People()
    {System.out.println("People Constructor");}
}
class Student extends People //subclass
{
    Student()
    {
        super(); //call super class constructor
        System.out.println("Student Constructor");
    }
}

class TestSPerson
{
    public static void main(String[] args)
    {
        Student s = new Student();
    }
}
```



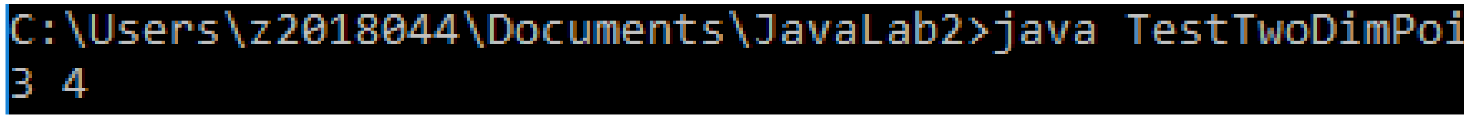
```
C:\Users\z2018044\Documents\JavaLab2>java TestSPerson
People Constructor
Student Constructor
```

Note: super() not needed as no values are passed

Subclass Constructors 1 (no `super` used)

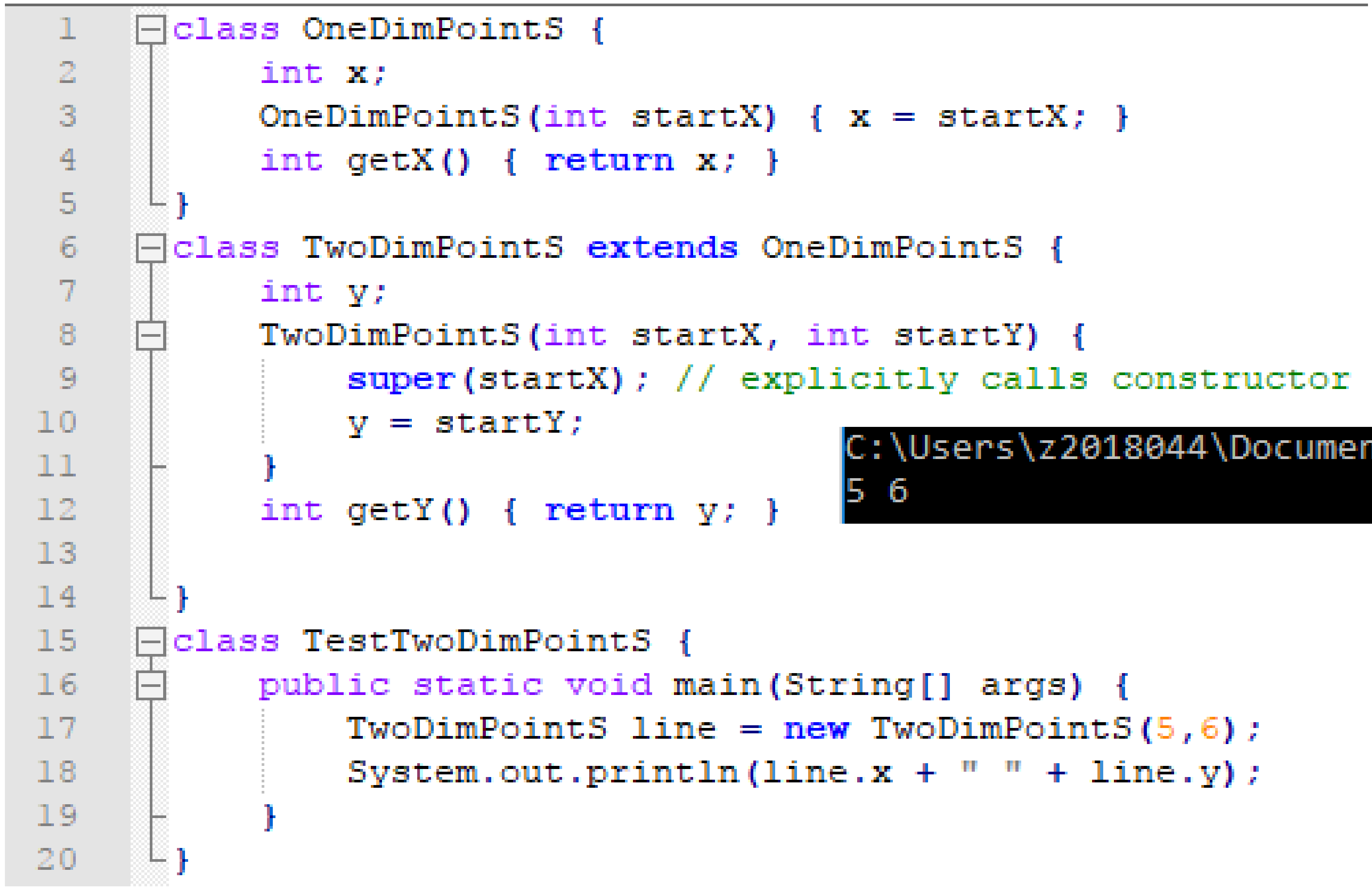


```java
class OneDimPoint {
    int x;
    OneDimPoint() { x = 3; } //super constructor
    int getX() { return x; }
}
class TwoDimPoint extends OneDimPoint {
    int y;
    TwoDimPoint() { y = 4; } // automatically calls
                             // OneDimPoint() first
    int getY() { return y; }
}
class TestTwoDimPoint {
    public static void main(String[] args) {
        TwoDimPoint line = new TwoDimPoint();
        System.out.println(line.x + " " + line.y);
    }
}
```

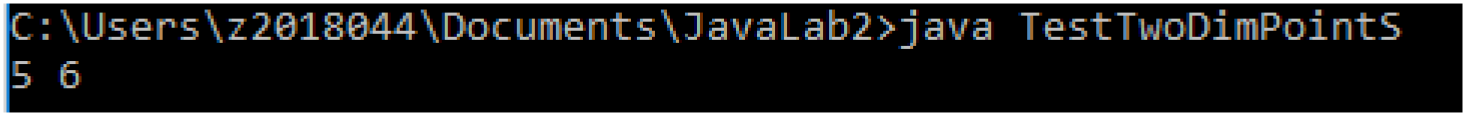


```
C:\Users\z2018044\Documents\JavaLab2>java TestTwoDimPoi
3 4
```

Subclass Constructors (`super` used)



```java
class OneDimPointS {
    int x;
    OneDimPointS(int startX) { x = startX; }
    int getX() { return x; }
}
class TwoDimPointS extends OneDimPointS {
    int y;
    TwoDimPointS(int startX, int startY) {
        super(startX); // explicitly calls constructor
        y = startY;
    }
    int getY() { return y; }

}
class TestTwoDimPointS {
    public static void main(String[] args) {
        TwoDimPointS line = new TwoDimPointS(5,6);
        System.out.println(line.x + " " + line.y);
    }
}
```

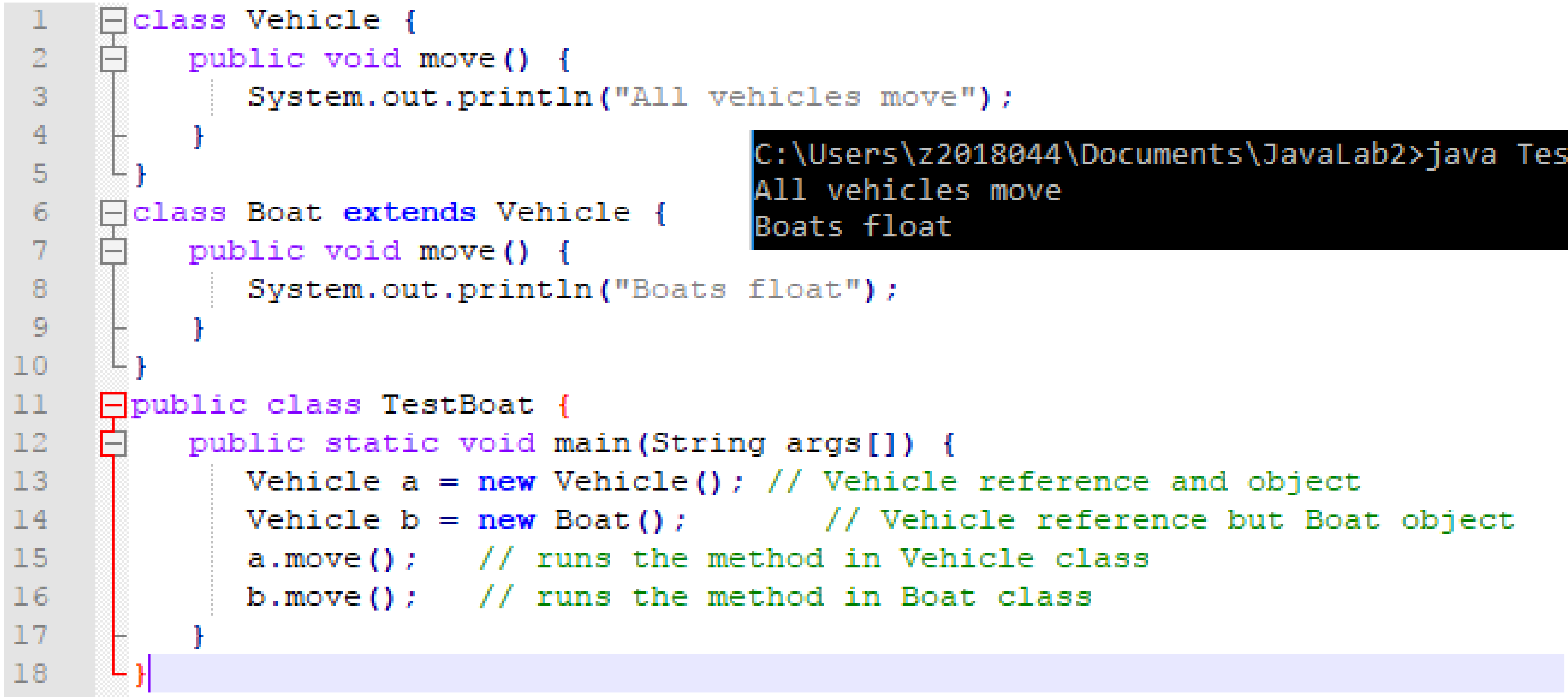


```
C:\Users\z2018044\Documents\JavaLab2>java TestTwoDimPointS
5 6
```

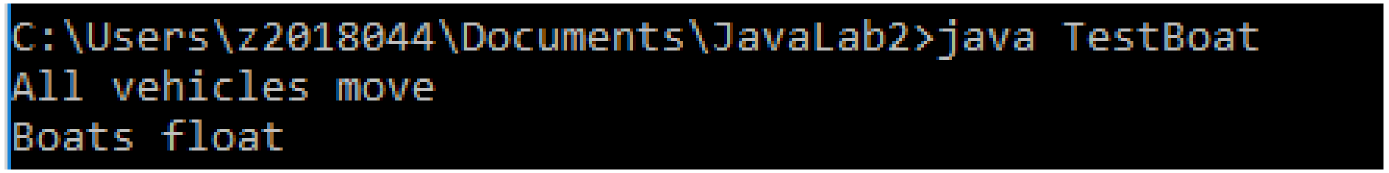
Overriding

- If a class inherits a method from its superclass, it has an option to override it (provided that it is not marked `final`)
- If methods are rewritten with the same name they replace the original method at compile time
- Also needs the same argument list and return type
- A method declared final or static cannot be overridden.
- Constructors cannot be overridden

Example Overriding

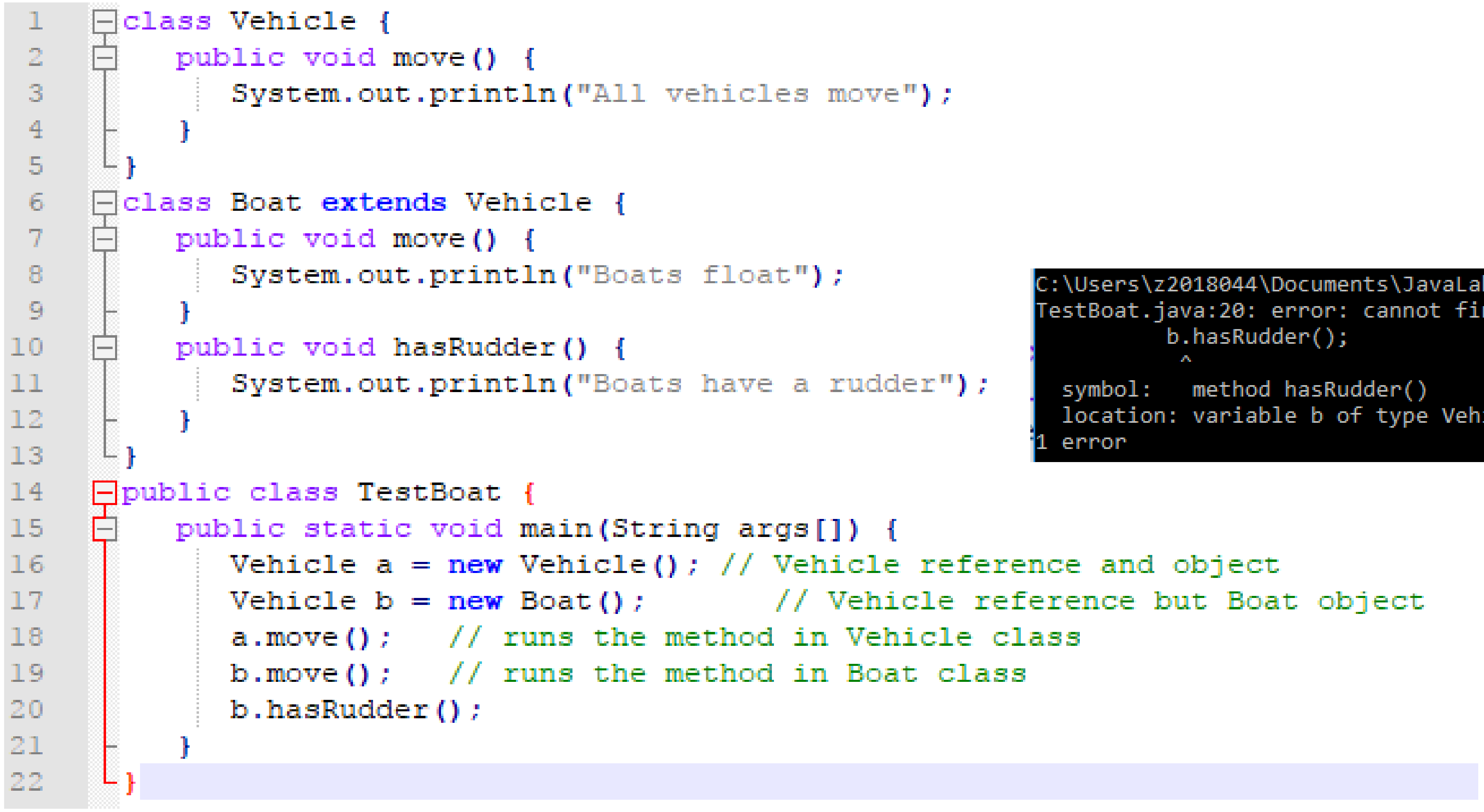


```java
class Vehicle {
    public void move() {
        System.out.println("All vehicles move");
    }
}
class Boat extends Vehicle {
    public void move() {
        System.out.println("Boats float");
    }
}
public class TestBoat {
    public static void main(String args[]) {
        Vehicle a = new Vehicle(); // Vehicle reference and object
        Vehicle b = new Boat();        // Vehicle reference but Boat object
        a.move();   // runs the method in Vehicle class
        b.move();   // runs the method in Boat class
    }
}
```

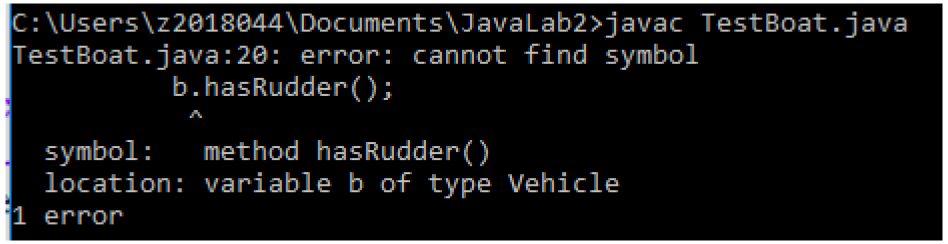


```
C:\Users\z2018044\Documents\JavaLab2>java TestBoat
All vehicles move
Boats float
```

..but this is restrictive, cannot add new methods

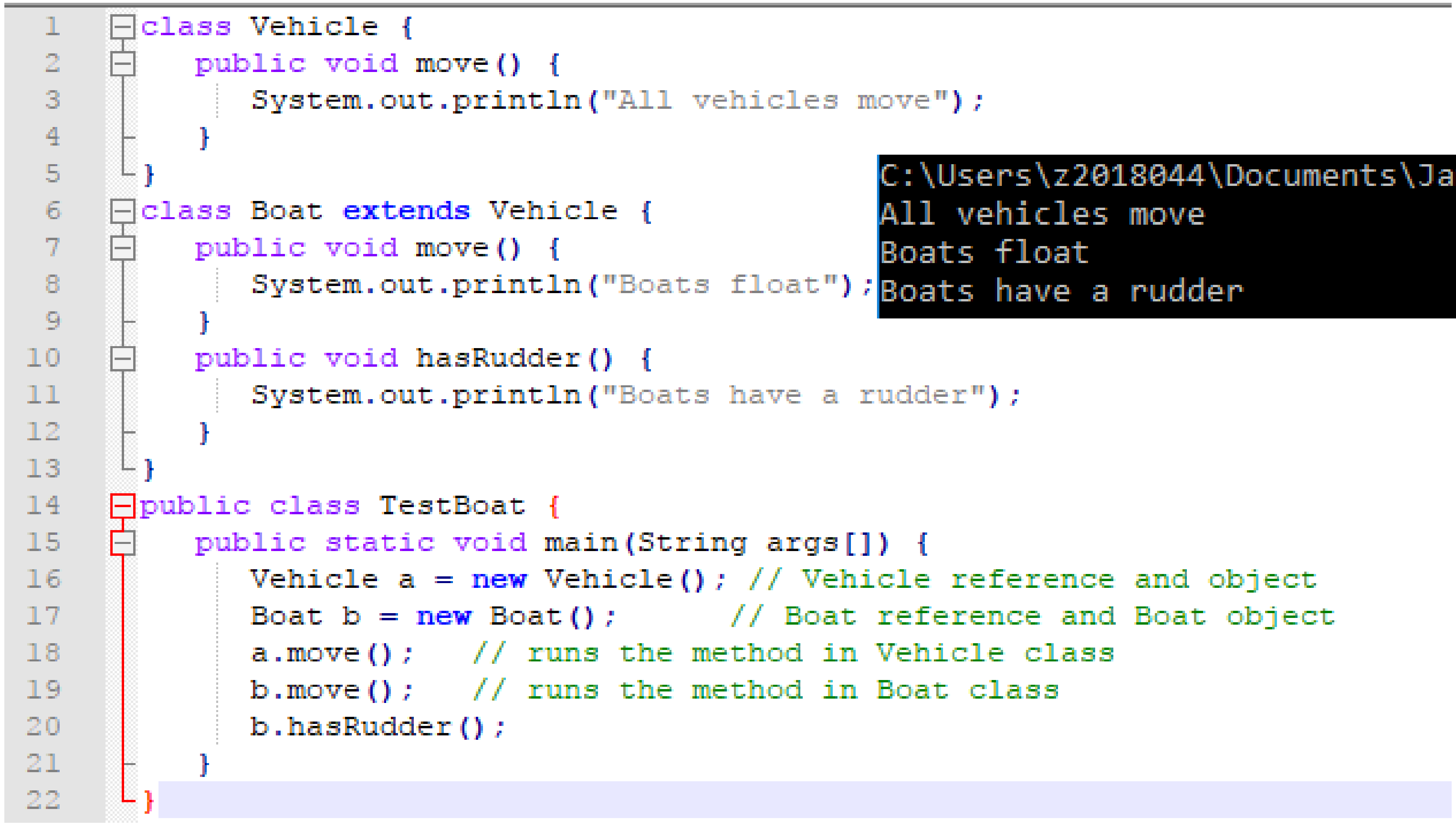


```
class Vehicle {
    public void move() {
        System.out.println("All vehicles move");
    }
}
class Boat extends Vehicle {
    public void move() {
        System.out.println("Boats float");
    }
    public void hasRudder() {
        System.out.println("Boats have a rudder");
    }
}
public class TestBoat {
    public static void main(String args[]) {
        Vehicle a = new Vehicle(); // Vehicle reference and object
        Vehicle b = new Boat();       // Vehicle reference but Boat object
        a.move();   // runs the method in Vehicle class
        b.move();   // runs the method in Boat class
        b.hasRudder();
    }
}
```

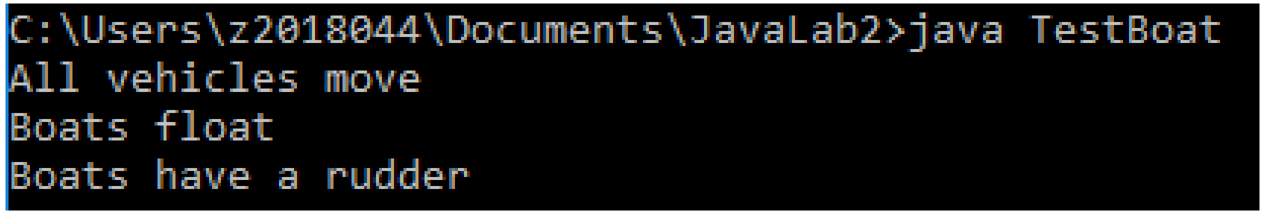


```
C:\Users\z2018044\Documents\JavaLab2>javac TestBoat.java
TestBoat.java:20: error: cannot find symbol
          b.hasRudder();
           ^
  symbol:   method hasRudder()
  location: variable b of type Vehicle
1 error
```

Solution



```java
class Vehicle {
    public void move() {
        System.out.println("All vehicles move");
    }
}
class Boat extends Vehicle {
    public void move() {
        System.out.println("Boats float");
    }
    public void hasRudder() {
        System.out.println("Boats have a rudder");
    }
}
public class TestBoat {
    public static void main(String args[]) {
        Vehicle a = new Vehicle(); // Vehicle reference and object
        Boat b = new Boat();      // Boat reference and Boat object
        a.move();   // runs the method in Vehicle class
        b.move();   // runs the method in Boat class
        b.hasRudder();
    }
}
```



```
C:\Users\z2018044\Documents\JavaLab2>java TestBoat
All vehicles move
Boats float
Boats have a rudder
```