CS220 — Software Development II

J Maraist

September 18, 2017

Outline

Contents

1 Introduction 1
2 JUnit and test-driven development 3
3 Classes, objects, inheritance and notation 13
4 Arrays 25
   4.1 An ADT for resizable lists ................................. 34

Course notes for CS220 are maintained in this file. They will be uploaded after classes, about once per week.

1 Introduction

The Software Design sequence

• Design algorithms
  – High-level problem-solving skills

• Implement algorithms as programs
  – Java - A modern programming language
  – Organize data and instructions
  – In both algorithms and programs, we must use low-level and precise logic
    * No ambiguity allowed
  – Debugging and testing

• Understand what programs will do

• Communicate technical information about your programs

• Learn how to operate as a technical professional
Software Design I

The six things a program can do

1. Get input
2. Give output
3. Do arithmetic
4. Update a stored value
5. Test a condition, and select an alternative
6. Repeat a group of actions

Four ways Java will help you organize your work

1. Grouping related data together
2. Defining sequences of operations
3. Associating data with operations relevant to the particular data
4. Naming these groups, sequences and associations for easy and repeated use

Software Design II

- A deeper look at inheritance and object-oriented design
- Exceptions
- Recursion
- Linear (list) data structures
- Describing and tracing the effects of programs
- Programming with files and directories
- Multi-dimensional arrays
- Debugging and testing
  - Knowing what "correct" means, and how to tell if your code meets that standard
  - This class will have a strong focus on test-driven development
- Skills-based class
- Many things we learn will build on what we’ve already studied
  - Including all of Software Design I
- Practice is essential
  - Expect to work on CS220 every day
Assignments
Each assignment is…

• Posted to the course website

• Submitted through AutoLab
  – https://euryale.cs.uwlax.edu/courses/cs220-fa17-jmaraist/
  – Remember the campus VPN vpn.uwlax.edu

• You should have received an email with account information on Monday
  – Check your spam filter
  – If you registered on Friday or after, I may not have your info
  – Email by 2pm if you still need one, and I’ll make additional accounts this afternoon

• Tomorrow’s lab will step you through a first AutoLab submission

Textbooks
There is no required text for this class, but you may be happier with a reference of some sort and/or a source of practice exercises. Some options:

• The CS120 online book, Programming in Java, zyBooks
  – Can print sections/chapters as well
  – Subscribe:
    * Sign up at zyBooks.com
    * Enter zyBook code: UWLAXCS220MaraistFall2017
    * Click Subscribe

• Java: A Beginner’s Guide, Herbert Schildt, Oracle Press
  – Several faculty recommend this book as a second reference for CS120

• O’Reilly has been reputable for reference books

2 JUnit and test-driven development

JUnit

• For specifying and running functional tests in Java

• A separate test for every method
  – Use Java annotations to mark the test

• The JUnit executable finds test methods, runs them, reports the results

• Eclipse will alert you to test failures
JUnit example test subject
From the JUnit wiki:

• A simple class

```java
public class Calculator {
    public int evaluate(final String expression) {
        int sum = 0;
        for (String summand: expression.split("\\+"))
            sum += Integer.valueOf(summand);
        return sum;
    }
}
```

JUnit example test class

• Test `Calculator` with

```java
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class CalculatorTest {
    @Test public void evaluatesExpression() {
        final Calculator calculator = new Calculator();
        final int sum = calculator.evaluate("1+2+3");
        assertEquals(6, sum);
    }
}
```

• The `@Test` annotation: how JUnit finds tests
  

• Method takes no parameters

• The class holding the tests has a zero-argument constructor
  
  – Which is implicitly present if we give no constructor at all

• Assertions
  
  – Methods provided by JUnit for asserting things which should be true
  
  
  – Parameter order: description, expected value, then actual value
    
    * Always give the description!
  
  – For real-valued tests, an additional `tolerance` parameter `delta`
  
  – Pointer equality vs. `equals`
JUnit Assert JavaDoc

org.junit

Class Assert

java.lang.Object
extends Object

public class Assert
extends Object

A set of assertion methods useful for writing tests. Only failed assertions are recorded. These methods can be used directly: Assert.assertEquals(...), however, they read better if they are referenced through static import:

```
import static org.junit.Assert.*;
...
assertEquals(...);
```

JUnit Assert methods

<table>
<thead>
<tr>
<th>static void</th>
<th>assertEquals(String message, double expected, double actual, double delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asserts that two doubles are equal to within a positive delta.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static void</th>
<th>assertEquals(String message, float expected, float actual, float delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asserts that two floats are equal to within a positive delta.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static void</th>
<th>assertEquals(String message, long expected, long actual)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asserts that two longs are equal.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static void</th>
<th>assertEquals(String message, Object[] expecteds, Object[] actuals)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Deprecated. use assertArrayEquals</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static void</th>
<th>assertEquals(String message, Object expected, Object actual)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asserts that two objects are equal.</td>
</tr>
</tbody>
</table>

JUnit Assert methods

<table>
<thead>
<tr>
<th>static void</th>
<th>assertNull(Object object)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asserts that an object is null.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static void</th>
<th>assertNull(String message, Object object)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asserts that an object is null.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static void</th>
<th>assertSame(Object expected, Object actual)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asserts that two objects refer to the same object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static void</th>
<th>assertSame(String message, Object expected, Object actual)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asserts that two objects refer to the same object.</td>
</tr>
</tbody>
</table>

JUnit and exceptions
Can also specify tests which we expect to fail
• The Java API says that `ArrayList.get(0)` will throw an `IndexOutOfBoundsException` when the list is empty.

• We verify this behavior with:

```java
@Test(expected = IndexOutOfBoundsException.class)
public void empty() {
    new ArrayList<Object>().get(0);
}
```

• Documented on Test JavaDoc page [junit.org/junit4/javadoc/latest/org/junit/Test.html](http://junit.org/junit4/javadoc/latest/org/junit/Test.html)

**ArrayList.get exceptions**

```java
public E get(int index)
    Returns the element at the specified position in this list.

    Specified by:
    get in interface List<E>

    Specified by:
    get in class AbstractList<E>

    Parameters:
    index - index of the element to return

    Returns:
    the element at the specified position in this list

    Throws:
    IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())
```

**Test annotation arguments**

From [junit.org/junit4/javadoc/latest/org/junit/Test.html](http://junit.org/junit4/javadoc/latest/org/junit/Test.html),
JUnit/AutoLab does not replace your other debugging skills!

- Use the main routine to dig deeper into your code

Trouble in StringCheckerTests

@Test public void
testGetBasisSingle() {
    final String[] basis =
        buildSingle().getBasis();
    assertEquals(
        "Only one member of the basis",
        1, basis.length);
    assertTrue(
        "Member of basis is \\
        + SINGLE + \\
        ,
        basis[0].equals(SINGLE));
}

- Normally would not want StringBuilder to refer to StringBuilderTests
  - But this is just temporary
  - Will clear main, use again for the next problem

Investigate with StringChecker.main

public static void main() {
    final String[] basis =
        StringBuilderTests.buildSingle()
            .getBasis();
System.out.println(
    "Only one member of the basis");
System.out.println(basis.length);

System.out.println(
    "Member of basis is \\
    " + StringBuilderTests.SINGLE
    + "\\n",
System.out.println(basis[0]);
)

• Shows us exactly what is happen in the test
  – Can add further print statements to methods
  – Just remove them/comment them out when the problem is fixed

**Test-driven development**
For class project we will adopt the discipline of *test-driven development* (TDD)

• When coding, repeat the following steps:
  1. Write a (failing) test case
  2. Get it to compile
  3. Get it to pass
  4. Simplify and remove duplication

• Once we have a test passing, we make sure that it *keeps on passing*

**TDD Example**

• Working on a financial application
• Need a class to make decisions as to whether certain commodities should be traded
  – A very mathematical question, so we’ll need to support various operations
• Specifically, we need to calculate the first statistical moment about a particular point
  – Don’t have a method for it yet
  – But our stats experts gave us a simple example for this test:

```java
@Test public void testFirstMoment() { final InstrumentCalculator calc = new InstrumentCalculator();
    calc.addElement(1.0); calc.addElement(2.0); assertEquals(
    "First moment about 2.0 for {1.0,2.0} within tolerance",
    -0.5, calc.firstMomentAbout(2.0), TOLERANCE); }
```
Making testFirstMoment compile

• This won’t even compile right now
  – We might already have a class InstrumentCalculator, but we haven’t written firstMomentAbout yet!
  – For this example, let’s assume we do have the class, with a method getElements()

• So we add a stub for the method, without trying to implement it
  – We could make it return an absurd value

    public class InstrumentCalculator { // ... keeping what’s already here
        public double firstMomentAbout(final double point) { return Double.NaN; } }

  – In our projects, we’ll usually throw an exception

    public double firstMomentAbout(final double point) { throw new UnsupportedOperationException(); }

• Whichever sort of stub we make, the test will now compile
  – But it will not yet pass

Making testFirstMoment pass

• The algorithm for calculating the first moment is standard — we look it up and implement it

    public double firstMomentAbout(final double point) { double numerator = 0.0; for(final double element : getElements()) {
        numerator += element - point;
    } return numerator / elements.size(); }

• Very often, a single test will correspond to much less code, or to only a small part of an algorithm

Write another failing test case

• The code we just added makes one test pass, but it’s not hard to conceive of cases which will fail
• There’s a division in the algorithm: are we safe against division by zero?
  – And what should happen when we call firstMomentAbout() with an empty data set?
  – Let’s say that the specification calls for an InvalidBasisException

• Write another test for this case!

    @Test(expected = InvalidBasisException.class) public void testEmptyFirstMoment() { new InstrumentCalculator().firstMomentAbout(0.0); fail("Expected InvalidBasisException"); }

    – fail is another JUnit method, like the assert methods, but never succeeding
Making `testEmptyFirstMoment` compile

What do we need to do to make `testEmptyFirstMoment` compile?

- If `InvalidBasisException` is **not already** part of `InstrumentCalculator`'s package, we must create it
- Otherwise it compiles

Making `testEmptyFirstMoment` pass

- We'll need to throw an `InvalidBasisException` when there are zero elements
- So a revised `firstMomentAbout`:

```java
public double firstMomentAbout(final double point) { if
        (getElements().isEmpty()) { throw new InvalidBasisException(); }

    double numerator = 0.0; for(final double element :
        getElements()) { numerator += element - point; } return
        numerator / elements.size(); }
```

- Now the test passes!

Write a failing test case

- Our next task is to write a routine for the second statistical moment about a point.
- So we write a test for this case. Our stats experts again gave us a simple example:

```java
@Test public void testSecondMoment() { final InstrumentCalculator
        calc = new InstrumentCalculator(); calc.addElement(1.0);
        calc.addElement(2.0); assertEquals( "Second moment about 2.0
        for {1.0,2.0} within tolerance", 0.5,
        calc.secondMomentAbout(2.0), TOLERANCE); }
```

Making `testSecondMoment` compile

- The problem is again that we do not define the method we are now testing
- And again we make it compile by adding a vacuous definition of the method. We'll just copy `firstMomentAbout` and change the name:

```java
public double secondMomentAbout(final double point) { if
        (getElements().isEmpty()) { throw new InvalidBasisException(); }

    double numerator = 0.0; for(final double element :
        getElements()) { numerator += element - point; } return
        numerator / elements.size(); }
```
Making testSecondMoment pass

- Unsurprisingly, the code for the first moment does not satisfy the second moment’s test!
- But the algorithm for the second moment is very similar, and we only need to make one change: from

  \[ \text{numerator} \text{+=} \text{element} - \text{point}; \]

  to

  \[ \text{numerator} \text{+= Math.pow(element - point, 2.0);} \]

- And now it passes!

Remove duplication

- This time around there’s definitely duplication — we have two methods that are almost completely identical!
- In fact, the algorithm for any of the statistical moments has only the same variation that we see here
- So the best way to remove this duplication is with a more general private method \text{nthMomentAbout} which the others call

  \[
  \text{public double nthMomentAbout(final double point, final double n)}
  \text{ \{ if (getElements().isEmpty()) \{ throw new InvalidBasisException(); \\}}
  \text{ \double numerator = 0.0; for(final double element : getElements()) \{ numerator += Math.pow(element - point, n); \\} return numerator / elements.size(); \}}
  \]

  \[
  \text{public double firstMomentAbout(final double point) \{ return nthMomentAbout(point, 1.0); \}}
  \]

  \[
  \text{public double secondMomentAbout(double point) \{ return nthMomentAbout(point, 2.0); \}}
  \]

- We already have tests in place, so we can be confident in this change

It’s OK to duplicate!

- This example seems draconian
  - Adding methods that we know are wrong!
  - Copying a method outright!
- But the point of TDD is that we are freed from worrying about more than one thing at a time
  - We might be setting up a test
– Or we might be writing code for a new feature, but never at the same time as setting up a test
– Or we might be refactoring away some horrible duplication, but never at the same time as setting up a test or writing new code
– Do one thing at a time, and do it right

• The tests we build up make later changes and additions much less risky

How we’ll learn TDD

• How does a specification become a series of tests?
  – Literally, we go sentence by sentence, translating the entire thing into a set of tests

• In the first two projects
  – I’ll give you the tests for each step
  – You’ll submit code making those tests — and only those tests — pass

• Over the semester, you’ll take responsibility for both the test and the primary code

The various files

• Three different roles, three different (sets of) classes and files
• All must compile together!

Interface

• Assignments usually specify one or more Java interfaces
  – Sec. 12.12 of the zyBook, or Ch. 8 of Schildt
• The methods I’ll expect your code to support
  – You should not edit the given interfaces!

Implementation

• Your job is to build these
• Just like what you’d write for CS120 projects

Test classes

• Makes sure that your interface methods perform as specified
• For early projects, given for each step
  – Run by AutoLab, available for you to run yourself
  – See the Testing this assignment paragraphs
• For later projects, you will write the tests yourself
3 Classes, objects, inheritance and notation

Real-life objects
This object is commonplace, and yet complicated

• It possesses some state
  – Including its current location, gear, current speed
• It has some behaviors
  – Like moving, accelerating, braking
• It interacts with other objects
  – Like the road, other cars, trees, people
• It is made of other objects
  – Like the engine, seats, tires, radio

Most of us can use it as a black box

• Don’t need to understand how it works
• Just need to know what we can do with it

Software objects

Definition
A software object is an entity in a program that possesses state (attributes) and behavior (actions).

• May interact with other objects
• May be composed of other objects
• Can be treated as a black box

Definition
Object-oriented programming (OOP) is a programming paradigm that uses the concept of objects to model entities. A program consists of interactions between some number of objects.
Object instances & classes

For our programs:

- **We do** want to be able to use many objects (which may be similar or different).
- **We don’t** want to have to rewrite lots of code.

In Java, we can achieve this by:

- Providing a template (blueprints) for all objects of the same type (group or kind) in a **class**, which specifies
  - attributes (data members), or properties of an object, and
  - behavior (actions) that an object can do
- Creating separate **instances** (objects) of that class to use in our programs

Class diagrams

**Definition**

A **class diagram** describes a class and how it can be used properly.

- Sketch of attributes and behaviors for objects of that type
- No details about *how* it works

<table>
<thead>
<tr>
<th>Class Name</th>
<th>attributes(instance variables)</th>
<th>operations(methods)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>- String makeModel - int mileage</td>
<td>«constructor»: Car(String, int) {0.5em} «update»: void setMakeModel(String) + void setMileage(int) {0.5em} «query»: String getMakeModel() + int getMileage()</td>
</tr>
</tbody>
</table>

Class diagrams - attributes

- The **Car** class has two simple **attributes** — used to store information about an individual car.
- Each different Car object (instance) in a program has its *own* attributes (instance variables).
- In Java, attributes correspond to fields
Class diagrams - private attributes

What do the minus signs mean?

- Both of the Car class instance variables have private access visibility.
  - Information only accessible from within the class

- In Java, private attributes correspond to private fields

- How can such private variables be read or written?

Class diagrams - instance methods

Instance Methods

- Update methods (Mutators)
  - Change something about the state of an object
  - Often void methods: do an action but don’t give output

- Query methods (Accessors)
  - Tell us something about the state of the object
  - Methods which are not void: give information back

Class diagrams - public methods

What does the plus sign mean?

- These methods all have public access
- Can be used outside of the class (from other code)
- Allows programmer to control how objects get modified and what information is revealed
Class diagrams - constructors

- Used to create an object
- Has the same name as the Class
- Has no (explicit) return type
- Almost always public (Why?)

The constructor here takes two parameters as input

Implementing the class

Creating and using objects in a program

- Creation:

```java
// GENERAL
final TYPE objectRef = new CONSTRUCTORNAME(params);

// CONCRETE
final Car myCivic = new Car("Honda Civic", 214118);
```

- Ask the object to perform an action:

```java
// GENERAL
objectRef.methodName(params);

// CONCRETE
myCivic.getMakeModel();
myCivic.setMileage(myCivic.getMileage() + 1);
```
Object-oriented design

Now that we have our `Car` class, we can build more complex classes that use cars.

- Let’s create a `CarLot` that tracks cars (such as for a used car dealership)

<table>
<thead>
<tr>
<th>CarLot</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Car <code>car1</code> - Car <code>car2</code></td>
<td>- String <code>makeModel</code> - int <code>mileage</code></td>
</tr>
<tr>
<td><code>constructor</code> +</td>
<td><code>constructor</code> +</td>
</tr>
<tr>
<td>CarLot()</td>
<td>Car(String, int) <code>setMakeModel</code>(String)</td>
</tr>
<tr>
<td><code>query</code></td>
<td><code>update</code></td>
</tr>
<tr>
<td>+ void <code>printCars</code>()</td>
<td>+ void <code>setMileage</code>(int)</td>
</tr>
</tbody>
</table>

- The `CarLot` is an *aggregate* class, made up of other objects
- Relationships between classes can be signified on UML
  - Diamond arrow is a "used by" relationship

Running a Java program

A `CarLot` will create `Car` objects, but what creates the `CarLot` itself? The solution is to use *static* methods (and variables).

- Content marked *static* is independent of any object instance
- Usually associated with the class itself

```java
public class CarLot {
    private Car car1;
    private Car car2;

    public static void main(String[] args) {
        CarLot myCarLot = new CarLot();
    }

    public CarLot() {
        car1 = new Car("Honda Civic", 214118);
        ...
    }
}
```

The main method

In Java, the `main` method has special significance

- Provides a point of entry for starting a program
  - Must be *public* and *static*
- *Any* class can have a `main` method
- Must have proper signature (including `String array param`)

```java
17
```
• In OO paradigm, main typically creates a top-level object and invokes a method which then takes over

```java
public class CarLot {
    ...
    public static void main(String[] args) {
        CarLot myCarLot = new CarLot();
        myCarLot.manage();
    }
    ...
    public void manage() {
        // Most of program functionality goes here
    }
}
```

**Adding more complexity**
Suppose the car dealership also sells commercial vehicles (trucks/vans) which have varying carrying capacities (e.g., 1/5/10 tons).

**One solution: Create a separate class**

<table>
<thead>
<tr>
<th>Car</th>
<th>Truck</th>
</tr>
</thead>
<tbody>
<tr>
<td>- String makeModel - int mileage</td>
<td>- String makeModel - int mileage - int capacity</td>
</tr>
<tr>
<td>«constructor»</td>
<td>«constructor» + Truck(String, int, int)[0.5em] «update»</td>
</tr>
</tbody>
</table>
| + Car(String, int)[0.5em] «update» | + void setMakeModel(String)
| + void setMakeModel(String) | + void setMileage(int) + void setCapacity(int)[0.5em] «query» + String
| + void setMileage(int) | + void setMileage(int) + void setCapacity(int)[0.5em] «query» + String
| «query» + String getMakeModel() + int getMileage() | + void getMakeModel() + int getMileage() + int getCapacity() |
Finding a better solution
Our first solution has some problems:

- Large amount of duplication
- Harder to write general-purpose code

Certain types of objects have things in common

- Cars/trucks/motorcycles
- Savings/checking/investment accounts

We should adjust our model to exploit these commonalities.

- Done via inheritance in Java

Inheritance

Inheritance is when one class (the subclass or child class) is based on another class (the super class or parent class), which the child class extends or modifies in some way.

- Superclass (or parent) contains similarities
- Subclass (or child) extends the parent
  - Inherits methods and variables from the parent
  - Can add more methods and variables or modify existing ones

Allows us to make our code simpler and more useful!

Inheritance in UML and Java

Inheritance can be represented in UML with arrows from children to parents

- Each child is a more specific kind of parent object
- Called an is-a relationship

```
public class Vehicle {
    // data and methods
}

public class Car extends Vehicle {
    // more data and methods
}

public class Truck extends Vehicle {
    // more data and methods
}
```
Class diagrams for inheritance

```
Vehicle
# String makeModel # int mileage
«constructor» + Vehicle(String, int) [0.5em] «update» + void setMakeModel(String) + void setMileage(int) [0.5em] «query» + String getMakeModel() + int getMileage()

Car
«constructor» + Car(String, int)

Truck
- int capacity
«constructor» + Truck(String, int, int) [0.5em] «update» + void setCapacity(int) [0.5em] «query» + int getCapacity()
```

The Vehicle superclass

The # indicates protected access

- Like public: accessible within any child subclass (and further descendants)
- Like private: not accessible from other classes

Variables and methods in Vehicle are inherited by all descendants

Implementing the Vehicle class

Other than the protected variables and the change of names, the code is identical to the Car class.

- Make sure to include comments in your code!

The Car subclass

```
Car
«constructor» + Car(String, int)
```

Every method and variable from Vehicle is inherited by Car

- Except for the constructor

A child’s constructor is responsible for calling the parent constructor.
• Called via super

        super();

• Must be the first action in the child constructor!
  – Call can be explicit or implicit (no arguments)

• Can also reference variables and methods in parent class

        super.methodName();
        super.variableIdentifier;

(Re-)Implementing the Car class

The Car constructor simply takes its inputs and "passes them up" to the Vehicle constructor via the super call. The Vehicle constructor is responsible for assigning the values to the instance variables.

    public class Car extends Vehicle {
        /**
         * Constructor for Car
         * @param makeModel The make and model of the Car
         * @param mileage The miles on the car
         */
        public Car(String makeModel, int mileage) {
            super(makeModel, mileage);
        }
    }

Implementing the Truck class

Inheritance structures

Objects can be part of an inheritance hierarchy, with multiple levels of ancestors and descendants.

• In Java, everything is descended from the Object class
Inheritance structures

Objects can be part of an inheritance hierarchy, with multiple levels of ancestors and descendants.

- In Java, everything is descended from the Object class

Type conformance

Every object conforms to the types of all its ancestors.

- In Java, everything conforms to the Object type

Conformance to interfaces

Interfaces specify methods without (normally) defining their bodies

- The names of the method
- The number of parameters each method has
- The type of each parameter
A class can implement one or more interfaces

- The class is then required to define a body for each interface method

Interfaces allow

- The description of what methods will be available to be separated from how they will be implemented
- Different implementations of the same method by different implementing classes

A vehicle interface

Every object conforms to both its ancestor classes and ancestor interfaces

```
public interface VInterf {
    public String getMakeModel();
    public String drive();
}
```

```
public class Vehicle implements VInterf {
    protected String makeModel;
    protected int mileage;

    // ...
}
```

Polymorphism

Polymorphism is the occurrence of something in several different forms.

- When you declare a variable, you assign its type
- As your program executes, a polymorphic variable can appear to change type, based upon the object it is currently referencing

```
Vehicle myVehicle;
myVehicle = new Car("Honda Civic", 214118);
myVehicle = new Truck("Ford F-150", 0, 2);
```

- For each assignment, the compiler checks if the type of the assigned object conforms to that of the variable.
- At runtime, the actual type of the object being referenced determines how it behaves. (Virtual method invocation)
Using polymorphism

Once we have a set of sub-classes of a common class, we can do things like create an array of objects of different subclass types

- Just as with variables, the array stores references to objects, not the objects themselves

```java
Vehicle v1, v2, v3, v4;
v1 = new Car("Honda Civic", 214118);
v2 = new Car("Saturn S-Series", 163518);
v3 = new Truck("Ford F-150", 1234, 2);
v4 = new Truck("Mack Truck", 300, 20);

Vehicle[] lot = new Vehicle[4];
lot[0] = v1;
lot[1] = v2;
lot[2] = v3;
lot[3] = v4;
```

- Each Vehicle variable stores a reference to an object that conforms to Vehicle

- Generic Vehicle array contains references to two different types of objects

Using polymorphism

Polymorphism allows us to write methods that work with multiple types of objects

```java
public class CarLot {
    private Vehicle[] lot;
    public CarLot() {
        lot = new Vehicle[4];
        lot[0] = new Car("Honda Civic", 214118);
        // ...
        printVehicles();
    }

    private void printVehicles() {
        for (int i = 0; i < lot.length; ++i) {
            System.out.println(lot[i].getMakeModel());
        }
    }
}
```

- Since every object referenced in the array conforms to Vehicle, they will all have access to the inherited getMakeModel method

Using polymorphism

For an array of type Vehicle, we can only use the objects in that array in ways that are possible for Vehicle objects

```java
public class CarLot {
    private Vehicle[] lot;
```
public CarLot() {
    lot = new Vehicle[4];
    lot[0] = new Car("Honda Civic", 214118);
    // ...
    printVehicles();
}

private void printVehicles() {
    for(int i=0; i<lot.length; ++i) {
        System.out.println(lot[i].getCapacity());
    }
}

• Error: Vehicle does not have a getCapacity method
  – Every Truck is necessarily a Vehicle, but
  – Not every Vehicle is necessarily a Truck

4 Arrays

Arrays

An array is a primitive data structure for storing multiple objects

• All elements of the array must have the same type
• The length of the array is fixed at its creation, and never changes
• Each position in the array stores a single element
• Each element is referenced by its index in the array

4 Arrays

Arrays

An array is a primitive data structure for storing multiple objects

• All elements of the array must have the same type
• The length of the array is fixed at its creation, and never changes
• Each position in the array stores a single element
• Each element is referenced by its index in the array

4 Arrays

Arrays

An array is a primitive data structure for storing multiple objects

• All elements of the array must have the same type
• The length of the array is fixed at its creation, and never changes
• Each position in the array stores a single element
• Each element is referenced by its index in the array

Basic syntax

• Declare an array (does not allocate memory):

    final dataType[] arrayName;

    – Alternative syntax

    final dataType arrayName[];
• Allocate memory for a previously declared array:

```
arrayName = new dataType[numberOfElements];
```

Size cannot be negative

• Store and retrieve values in array:

```
arrayName[index] = expression;  // Store value at index
arrayName[index];              // Retrieve value from index
```

• Access the length of an array:

```
arrayName.length
```

Not the same as the method call for String — `str.length()`

• One-liners for declaration, allocation, and initialization:

```
final dataType[] arrayName = new dataType[ numberOfElements ];
final dataType[] arrayName = { val1, val2, ..., valN };  // Initialize
```

Basic examples

• Declare an array (does not allocate memory):

```
double[] numbers;  /* Alternate: */ double numbers[];
```

• Allocate memory for a previously declared array:

```
numbers = new double[10];
```

• Store and retrieve values in array:

```
numbers[3] = 7.5;
System.out.print(numbers[3]);
```

• Access the length of an array:

```
numbers.length
```

• One-liners for declaration, allocation, and initialization:

```
double[] numbers = new double[10];
double[] numbers = { 1.5, 4.5, 7.5, ..., 15.2 };`
**Arrays of primitive types**

When using arrays, we need to ensure:

- Array variable is declared
- Memory is allocated for the array (using `new`)
- Contents of the array have been initialized

With primitive type:

```java
final int[] intArray = new int[5];
for(int i=0; i<intArray.length; ++i) {
    System.out.print(intArray[i] + "", ");
}
```

- Output:

```
0, 0, 0, 0, 0,
```

This works even though we skipped Step 3 – Java takes care of the initialization for us.

**Simpler iteration**

For loops make it easy to work with arrays

```java
final int[] intArray = new int[5];
for(final int j : intArray) {
    System.out.print(j + "", ");
}
```

**Arrays of objects**

We can also have arrays of complex type:

```java
final Car[] carArray = new Car[5];
for(int i=0; i<carArray.length; ++i) {
    System.out.println(carArray[i]);
}
```

- Output is

```
null
null
null
null
null
```

Why does this fail? Need to *initialize* array contents!

- Java doesn’t know how to initialize the objects
Example: all Civics

```java
final Car[] carArray = new Car[5];
// We need to initialize the objects in a sensible way
for(int i=0; i<carArray.length; ++i) {
    carArray[i] = new Car("Honda Civic", 1000 * i);
}
for(int i=0; i<carArray.length; ++i) {
    System.out.println(carArray[i]);
}
```

Arrays of objects

```java
final Person[] simpsons = new Person[3];
simpsons[0] = new Person("Homer", "D'oh!");
simpsons[1] = new Person("Flanders", "Okily Dokily!");
```

In an array of complex type (i.e., class), each element in the array stores a reference to an object of that class

- Does not store the object itself (just like a variable of complex type)
- We need to instantiate an object for each element of the array

Arrays and methods

```java
public static void main(String[] args) {
    int[] myArr = { 1, 2, 3 };  
    squareArray(myArr);
    System.out.println(myArr[2]);
}

public static void squareArray(int[] arr) {
    for (int i = 0; i < arr.length; ++i) {
        arr[i] = arr[i] * arr[i];
    }
}
```

When an array is passed to a method, only its reference is passed (just like objects)

- Any modifications that the method does to the array contents persist after the method ends
Example
When an array is passed to a method, only its reference is passed (just like objects)

- The update to arr does not change what myArr references

```java
public static void main(String[] args) {
    int[] myArr = { 1, 2, 3 };
    modifyArray(myArr);
    System.out.println(myArr[2]);
}

grow static void
modifyArray(int[] arr) {
    arr[0] = 7;
    arr = new int[3];
    arr[2] = 9;
}
```

Multi-Dimensional Arrays
In Java, arrays can be extended to more than one dimension.

- A one-dimensional array:

  ```java
  int[] arr1d = new int[6];
  arr1d[3] = 7;
  ```

- A two-dimensional array:

  ```java
  int[][] arr2d = new int[3][5];
  arr2d[1][2] = 4;
  ```

- Accessing dimensions:

  ```java
  int[][] matrix = new int[7][10];
  int numRows = matrix.length; // Returns 7
  int numCols = matrix[0].length; // Returns 10
  ```

```
[0] [1] [2] [3] [4] [5]
[ ] [ ] [ ] [ ] [ ] [7]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
```
Using multi-dimensional arrays

Multi-dimensional arrays are useful for storing data that has *multiple indices*

- That is, "keys" to look it up

For example, storing movie reviews across users

```java
final int numPeople = 3;
final int numMovies = 5;
final int[][] ratings =
    new int[numPeople][numMovies];

// ...

ratings[0][3] = 5;
```

<table>
<thead>
<tr>
<th>movie (2nd index)</th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>reviewer (1st index)</td>
<td>[0]</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>[2]</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Multi-dimensional arrays

... and on to higher dimensions

- A one-dimensional array

```java
int[] arr1d = new int[6];
arr1d[3] = 7;
```

- A two-dimensional array

```java
int[][] arr2d = new int[3][5];
arr2d[1][2] = 4;
```

- A three-dimensional array

```java
int[][][] arr3d = new int[2][2][4];
arr3d[0][1][2] = 6;
```

First index is like the page number of a notebook

- And so on
Using multi-dimensional arrays

Another example: hourly temperatures for a weather station over 3 years

```java
int years = 3;
int days = 365;
int hours = 24;
double[][][] temps =
    new double[years][days][hours];
```

Storing temperature of $-1.2$ for Year 2 of 3, January 01, at 12 noon:

```java
temps[1][0][12] = -1.2;
```

Using multi-dimensional arrays

Just as a single `for` loop is useful for manipulating a one-dimensional array, nested `for` loops are useful for manipulating an $n$-dimensional array

- One loop per dimension

```java
int[][] arr2d = new int[20][15];

for(int row=0; row<20; ++row) {
    for(int col=0; col<15; ++col) {
        final int n = arr2d[row][col];
        System.out.print(n + " ");
    }
    System.out.println();
}
```

- `row` loops over the first dimension
- `col` loops over the second dimension
- This code does work, but what is `wrong` with it?
Avoid "magic numbers" in code
Hard-coding values leads to fragile code

- Difficult to maintain, hard to debug

Arrays of arrays
A two-dimensional array is actually an array of arrays!

```java
// Allocate space for 10 references to int[]
final int[][] arr2d = new int[10][];

// Allocate space for each "row"
for(int i=0; i<arr2d.length; ++i) {
    arr2d[i] = new int[5];
}
```

- arr2d is a variable that contains a reference to an array
  - arr2d.length gives size of this array
  - arr2d[i] gives element at position i
- arr2d[i] stores a reference to another array
  - arr2d[i].length gives size of this other array
  - arr2d[i][j] gives element at position j in this other array

Ragged arrays
What happens if we make this change to the array builder?

```java
// Allocate space for 10 references to int[]
final int[][] arr2d = new int[10][];

for(int i=0; i<arr2d.length; ++i) {
    // Allocate space for each "row"
    arr2d[i] = new int[i+1];
}
```
Creating a ragged array (as opposed to a rectangular array)

Creating multi-dimensional arrays

- Creating a rectangular two-dimensional array:

  // Allocate all space for array at once
  int[][] matrix = new int[5][4]; // 5 rows and 4 columns

  // Shortcut initialization: 2d array with 2 rows and
  // 3 columns
  int[][] matrix = { { 3, 5, 6}, {2, 4, 7} };

- Creating a ragged two-dimensional array:

  // Allocate memory for each row separately
  int[][] matrix = new int[3][]; // 3 rows
  matrix[0] = new int[5];  // 5 columns in row 0
  matrix[1] = new int[3];  // 3 columns in row 1
  matrix[2] = new int[7];  // 7 columns in row 2

  // Shortcut init.: 2 rows with 2 and 4 cols, respectively
  int[][] matrix = { { 2, 4}, {7, 3, 5, 6} };

Writing some code

Assuming that the matrix in the skeleton below is initialized, write the code necessary to multiply every entry by scalar.

double[][] matrix;
// Assume matrix initialized here
double scalar = ...;

// Your code here...

Writing some more code

Write a public static non-void method named matrixContains that takes a 2-dimensional array of integers and an integer and returns true if the matrix contains that value, false otherwise.

public static boolean matrixContains(int[][] matrix,
                                      int value) {

4.1 An ADT for resizable lists

Memory in Java
For primitive arrays in Java, we have the following requirement:

- The length of the array is fixed at creation (it never changes)

Why?
Consider the following:

```java
int[] arr1 = new int[3];
int[] arr2 = new int[4];
arr1.resize(5); // NOTE: Not an actual method for arrays!
```

A workaround?
This limitation can be problematic, as we may not know up front how many things we need to store.

- Allocate extra space for every array to reduce the likelihood of running out of room
- Create an overflow array to store additional items if/when first array is filled
- Create and use a larger array that can store old items plus new ones

```java
final int[] array = new int[5];
// .. intermediate work

// What should newSize be?
final int[] newArray = new int[newSize];
for(int i=0; i<array.length; ++i) {
    newArray[i] = array[i];
}
array = newArray;
```
Resizing an array?
Resizing an array is a common thing to want to do. Can we make a method for it?

```java
public static void resizeArray(int[] array, int newSize) {
    final int[] newArray = new int[newSize];
    for(int i=0; i<array.length; ++i) {
        newArray[i] = array[i];
    }
    array = newArray;
}
```

But this won’t work

```java
int[] myArr = new int[20];
resizeArray(myArr, 30);
myArr[25] = 42; // Error here - index out of bounds
```

The method can change the contents of the array that `myArr` points to, but it is unable to change the memory location that `myArr` references.

Resizing an array: Take 2
Resizing an array is a common thing we might want to do, so let’s make a method for it.

```java
public static int[] resizeArray(int[] array, int newSize) {
    int[] newArray = new int[newSize];
    for(int i = 0; i < array.length; ++i) {
        newArray[i] = array[i];
    }
    return newArray;
}
```

Now we can do the following:

```java
final int[] myArr = new int[20];
myArr = resizeArray(myArr, 30);
myArr[25] = 42; // This works!
```

The method returns a reference to the new array, which can be used to update `myArr`.

Resizing an array: time for Take 3?
The following scenario still poses a problem:

```java
public static void main(String[] args) {
    int[] array = new int[5];
    populateList(array);
    printList(array);
}
```

```java
public static void populateList(int[] array) {
    // work ..
    array = resizeArray(array, 10);
}```
A better solution
Add a layer of abstraction between the array contents and those who need to use it!

All problems in computer science can be solved by another level of indirection. — David Wheeler

Abstract data types
An abstract data type is a model that defines data types in terms of their behavior (what can be done with it).

What this means for the user:

• Do need to know what they can do with the data type
• Don’t need know how the data type is implemented

What does this remind you of?
Example: the String class in Java

• We manipulate String objects using public methods
• We don’t need to know the underlying representation of the characters to use it (though it can be helpful!)

Defining an abstract data type
Let’s consider our basic needs for a resizeable array:

• Add items (at end or at specific position)
• Access items
• Get number of items
We can abstract these needs as a list

<table>
<thead>
<tr>
<th>MyList</th>
</tr>
</thead>
</table>
| «constructor» + MyList()  
| «update» + void add(String) + void add(String, int)  
| «query» + String get(int) + int size() |

Implementation

<table>
<thead>
<tr>
<th>MyList</th>
</tr>
</thead>
</table>
| «constructor» + MyList()  
| «update» + void add(String) + void add(String, int)  
| «query» + String get(int) + int size() |

```java
public class MyList {
    public MyList() { // ...
    }
    public void add(final String item) { // ...
    }
    public void add(final String item, final int index) { // ...
    }
    public String get(final int index) { // ...
    }
    public int size() { // ...
    }
}
```

First step — the empty list

The test

```java
@Test public void emptyList() {
    final MyList ml = new MyList();
    assertEquals(0, ml.size());
}
```
Making it pass

private String[]
    strings = new String[0];

public MyList() {}

public int size() {
    return strings.length;
}

One piece of actual stuff

The test

@Test public void appendingStuff() {
    final MyList
        ml = new MyList();
    ml.add("stuff");
    assertEquals
        (1, ml.size());
    assertEquals
        ("stuff",
            ml.get(0));
}

Making the test pass

private String[]
    strings = new String[0];
public void add(final String item) {
    final int oldLen = size();
    final String[] newStrings
        = new String[1+oldLen];
    for(int i=0; i<oldLen; ++i) {
        newStrings[i] = strings[i];
    }
    newStrings[oldLen] = item;
    strings = newStrings;
}

Reading the empty list

In fact we could strengthen the first test a bit:

@Test(expected=ArrayIndexOutOfBoundsException.class)
public void emptyList() {
    final MyList ml = new MyList();

assertEquals(0, ml.size());
ml.get(0);
}

Is there extra work for multiple appends?

The test
We can add to the appendingStuff test:

```java
lst.add("thing2");
assertEquals("List at length 2",
             2, lst.size());
lst.add("thing3");
assertEquals("List at length 3",
             3, lst.size());
assertEquals("Get what you give #2",
             "thing2", lst.get(1));
assertEquals("Get what you give #3",
             "thing3", lst.get(2));
```

It works as is!
Our add(String item) method works just fine for these additional tests

What about the other add method?

The test
We’ll start the same way we did with the appending add

```java
@Test public void insertingStuff() {
    final MyList lst = new MyList();
    lst.add("stuff", 0);
    assertEquals(1, lst.size());
    assertEquals("stuff", lst.get(0));
}
```

What about the other add method?

But what’s really important is that we can insert from either side

```java
@Test public void insertingLeft() {
    final MyList lst = new MyList();
    lst.add("thing1", 0);
```
The inserting add

- Remember the old array
- Set up the new array
- Old elements before index have the same position
- Old elements after index shift right
- The new element’s insertion point
- Tests pass!
- Tests for more than two elements?
for(int i=0; i<index; i++) {
    strings[i] = oldStrings[i];
}

for(int i=oldSize; i>index; i--) {
    strings[i] = oldStrings[i-1];
}

strings[0] = item;