Topics: Polymorphism review, 2-d arrays

Reading (JV): Sec 6.4

Polymorphism example

```
public class Vehicle {
  protected int plateNum;  //plate #
  private int numWheels;   //# of wheels
  public void writeOut() { . . . }
} //class Vehicle

public class Plane extends Vehicle {
  protected double wingSpan;
    public void writeOut() { . . . }
  public void doSomething()  { . . . }
} //class Plane

public class Transport {
  Vehicle[] v = new Vehicle[5];
  v[0] = new Vehicle();
  v[1] = new Plane();
  v[1].writeOut();    //?
  v[1].doSomething(); //?
}
```

Accessing methods and fields through polymorphic references

- For overridden methods, the ***type of the object*** determines which version of the method gets invoked
- For other methods and fields, the ***type of the reference*** determines what methods and fields can be accessed

`Object` class

- If a class is not explicitly defined to be the child of an existing class, it is assumed to be the child of the **`Object`** class. Therefore all classes are (ultimately) derived from the **`Object`** class.
- The class header **`class Room`** is the same as **`class Room extends Object`**

`abstract` class

- A placeholder in a class hierarchy that represents a generic concept
- Cannot be instantiated
- Modifier: **`abstract`**

  `public abstract class Geometry`
- Can contain abstract methods

  `public abstract double Area();`
- Subclasses of abstract classes will "fill out" these abstract methods

Two-dimensional (2-d) arrays

- A *table* of values (references)
- Declare and access using *two* index values
- In Java, a 2-d array is an array of arrays (array of objects)
  - The orientation (row, column) is how *we choose to* visualize (organize) the table
  - By convention, we use ***row-major*** 2-d arrays

Multi-dimensional arrays

- Can have as many dimensions as you want
- Each dimension has its own constant **`length`**
- Since each dimension is an array of array references, it can have its own value of **`length`** ⇒ a *ragged* array

Creating a 2-d array

- Declare a reference **`x`** for a 2-d integer array:
- Create a 2-by-3 integer array **`y`**:
- Create the following array:

```
2 4 6
8 1 3
```

Accessing a 2-d array

Given a reference x that points to a 2-d integer array. . .

- What is its height (# of rows)?
- What is **`x[0]`** ?
- What is the length of the first row?
- How to access last element in row 2?
- How to access last element in last row?