Object Oriented programming in Java

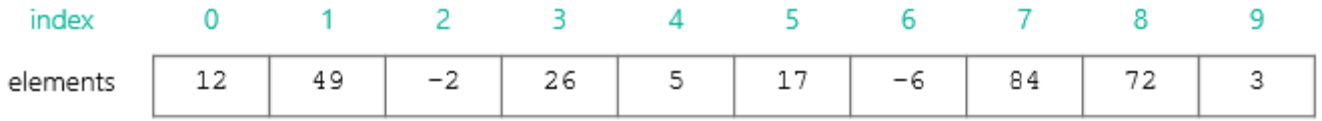
What is an Object?

- multiple pieces of data and the methods who act on this data under a single data type

- location for the data in the actual variable as a "reference“
- rely upon the methods provided by the object
- Null as analogous to zero for primitives, on objects

Arrays

- object that holds multiple pieces of the same type of data.
- Each element has an index




```
dataType[] arrayVariableName = new dataType[numberOfElementsToStore];
```

- When you first create an array it is filled with "zero values".
- Syntax to update the value stored in an index:

```
arrayName[index] = value;
```

- We can also create an Array and fill it with values in a single statement

```
dataType[] name = {dataValue1, dataValue2, dataValue3};
```

- Arrays and Methods

```
public static int[] myMethod(int[] a) {}
```

- "**IndexOutOfBoundsException**“: trying to access an index that does not exist, or beyond the capacity of the array.

Array attributes, methods and Arrays class

- Attribute that tells us the length of an array:

```
arrayVariableName.length // returns the capacity of the array
```

- Java provides a class for handling arrays, called Arrays :v

```
int[] b = new int[10];
System.out.println(Arrays.toString(a));
```

Always use Arrays.toString() to output the value of an array, otherwise you just get the addres in memory

More useful methods

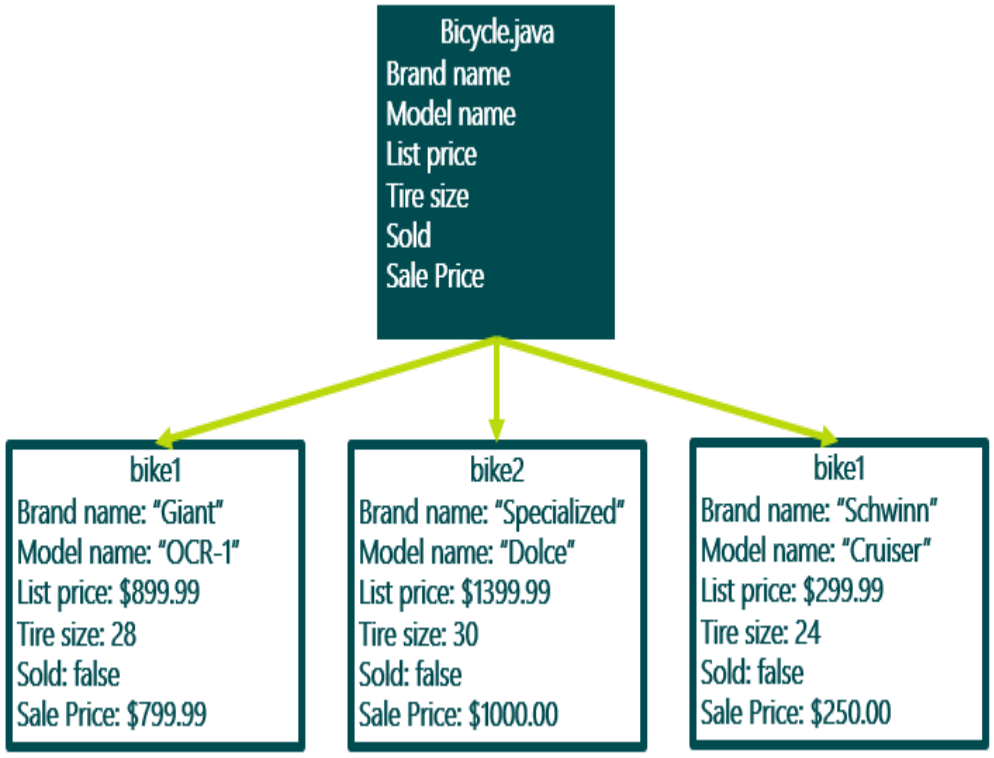
method	example	description
toString(array)	Arrays.toString(a)	returns a String representation of the array using square brackets and commas like so: [value, value, value]
equals(array1, array2)	Arrays.equal(a, b) OR a.equals(b)	returns true if the two arrays contain the same elements in the same order
fill(array, value)	Arrays.fill(a, 10)	fills every index of the array with a copy of the given value
copyOf(array, newLength)	Arrays.copyOf(a, 10)	creates a new array object with the given length and fills it with values in the same order as the original array. If there are left over indexes those are filled with the data type's zero value
sort(array)	Arrays.sort(a)	arranges the values in the array in sorted order from smallest to largest
binarySearch(array, value)	Arrays.binarySearch(a, 100)	returns the index of th given value in a **sorted** array. Will return a negative number if the value doesn't exist in the array.

Reference Semantics vs Value Semantics

- Primitives are stored directly in memory
  - when you copy a primitive variable the value is copied, leaving the original variable unaffected (Value semantics)

- Object variables store addresses, but the data is stored on a special space in memory
  - Any variable holding an object, stores a reference
  - When we copy these variables, we copy the address, resulting in both pointing to the same data. (reference semantics)

Creating Objects



- have to write a class that "defines" it.
  - this class does *not* have a main method
  - must instead be invoked by a different class that does have a main method.
- The class is a “mold”, the instance or the object invoked is a “particular individual of that design”

```
Bicycle bike1 = new Bicycle();
```

Creating instances on the main method

Creating Objects

```
public class Student {
   String name;
   int grad;
   int ID;
   double GPA;
   int abs;
   public Student(String name, int grad, int ID, double GPA, int abs) {
      this.name = name;
      this.grad = grad;
      this.ID = ID;
      this.GPA = GPA;
      this.abs = abs;
      }
      public boolean isGraduating() { return (GPA > 2.0 && abs < 10 && grad == 12);
  }
}
```

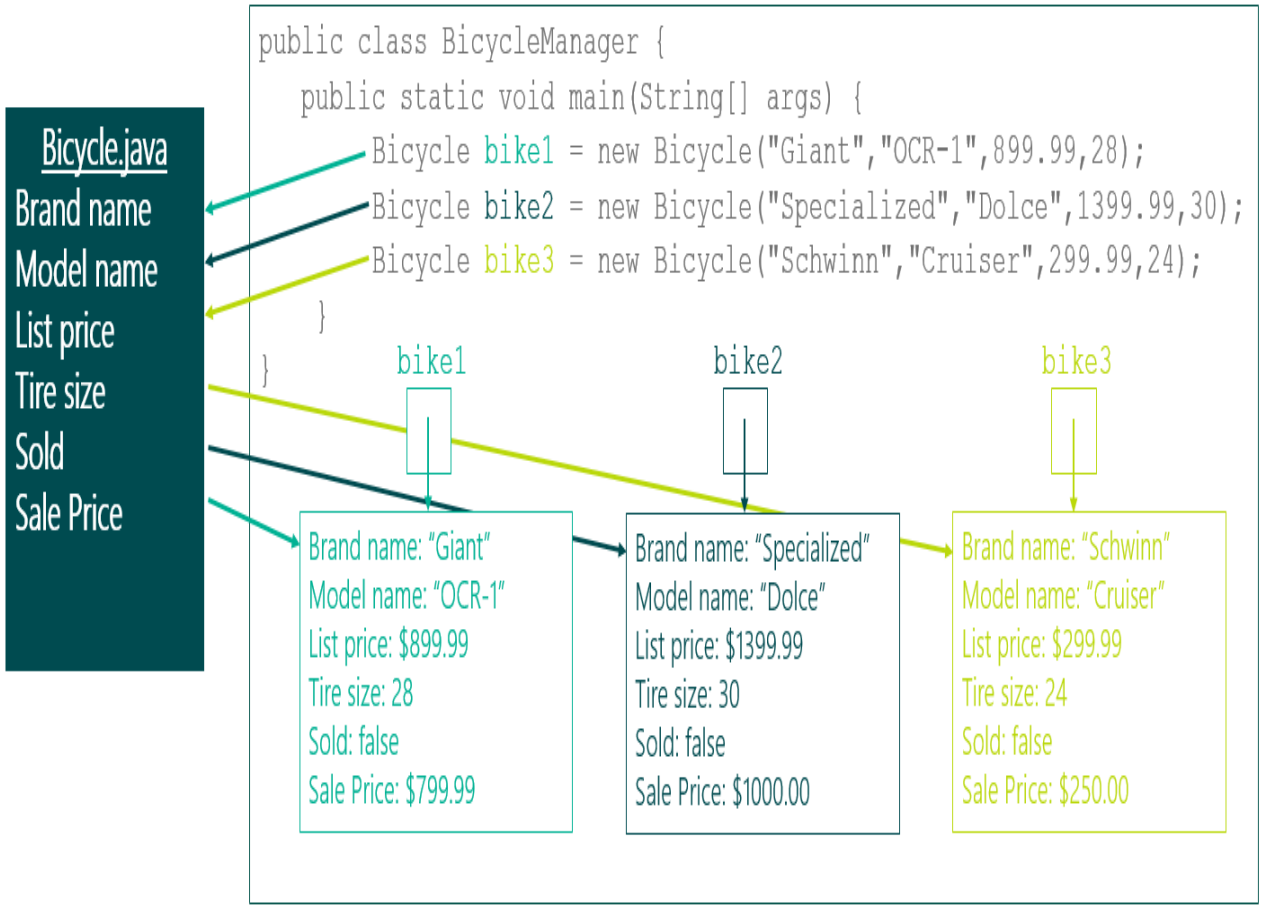
Class header

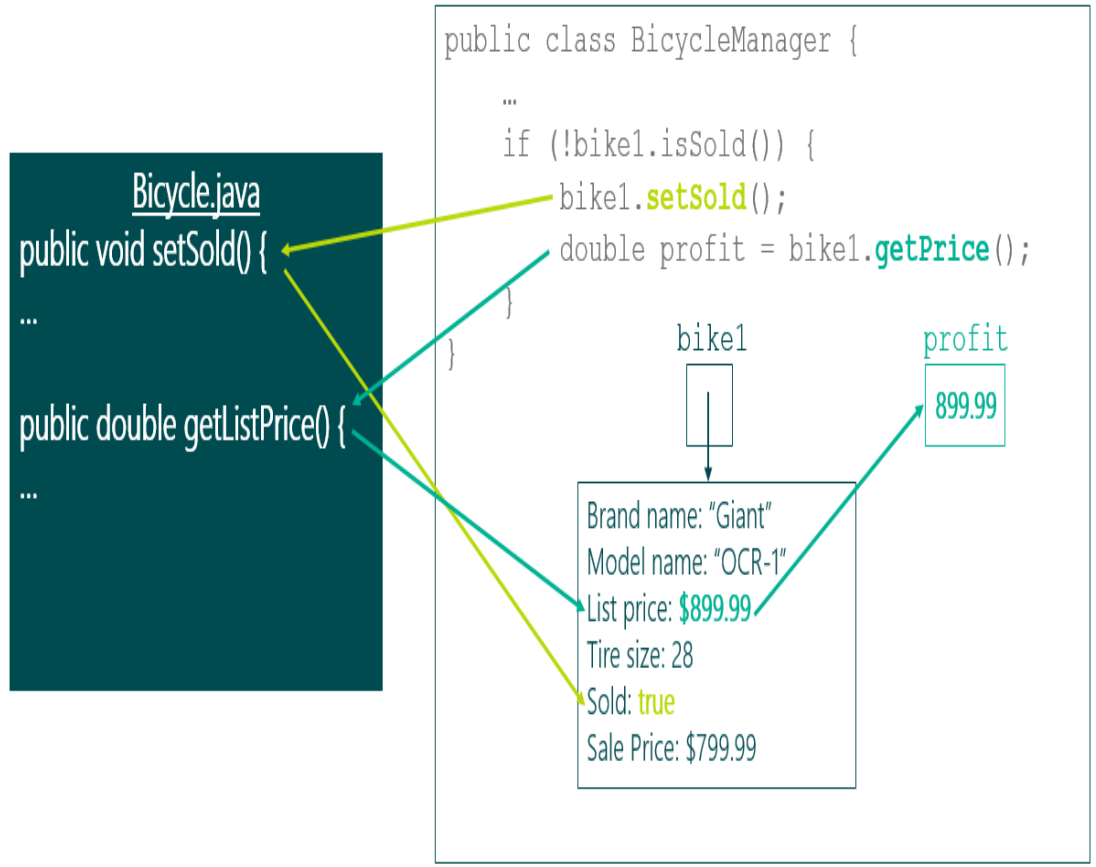
Attributes

Constructor

Methods

Manipulating objects in Client Class





State of an object, and encapsulation

- Collection of values that are stored within an object's fields. Fields can be public or private.
- Getters and Setters are used to see and manipulate the state, respectively.

```
public class Bicycle {
    private double listPrice = 899.99;
    private double salePrice = 599.99;
   // Updates the price of the bicycle
   public void setSale() {
      listPrice = salePrice;
   }
   public double getSalePrice(){
       return this.listPrice;
   }
}
```

Setter

Getter

It is good practice to set all fields to private, and access and modify them through getters and setters in client class, inside main method.

```
Bicycle myBike = new Bicycle();
//Updating listPrice value of myBike
myBike.setSale(1000.00);
```

Behavior of an Object

- Collection of methods within the object.
- Represent the different thing we want our object to be able to do

```
// marks the bicycle as sold and returns
// the asking price for the customer
// during a sale

public double makeReducedPriceSale() {
   isSold = true;
   return salePrice;
}

//Our object's methods can access the field in other objects of the same type.

public boolean equals(Bicycle other){
return this.brand.equals(other.brand) && this.model.equals(other.model);
 }
```

Constructors

- client class creates objects by using the new operator.
  - The new operator calls an object contructor.

```
Bicycle bike1 = new Bicycle("Giant","OCR-1",899.99,28);
```

- Contructors are special methods that creates instances and sets initial values.

```
public Bicycle (String myBrand, String
myModel, double myPrice) {
        this.brand = myBrand;
        this.model = myModel;
        this.price = myPrice;
 }
```

Has the same name as the name of the class

Does not specify return type, otherwise it would be a normal function

Default Contructors

- If we don’t specify a constructor, java creates one for us by default. It would look like:

```
public Bicycle() { }
```

- If we don’t specify constructor, then we cannot pass arguments to the new keyword in the client class:

```
Bicycle bike1 = new Bicycle();
```

Overloading Constructors

- Allow users decide what amount of data to specify when creating objects, and what to rely on default values.

- First create an initial constructor that initializes all fields.
- Create other constructors with desired amount of parameters.

```
public Bicycle(String brand, String model, double listPrice) {
        this(brand, model, listPrice, 28, false, listPrice);
 }
// this uses all default values for a new Bicycle
 public Bicycle() {
        this("", "", 0.0, 28, false, 0.0);
 }
```

Static Fields (class variable)

- Their values are not specific to a single object.
  - They are shared by all objects of the same class
  - All objects can mutate this value, and would be the same for any of them.

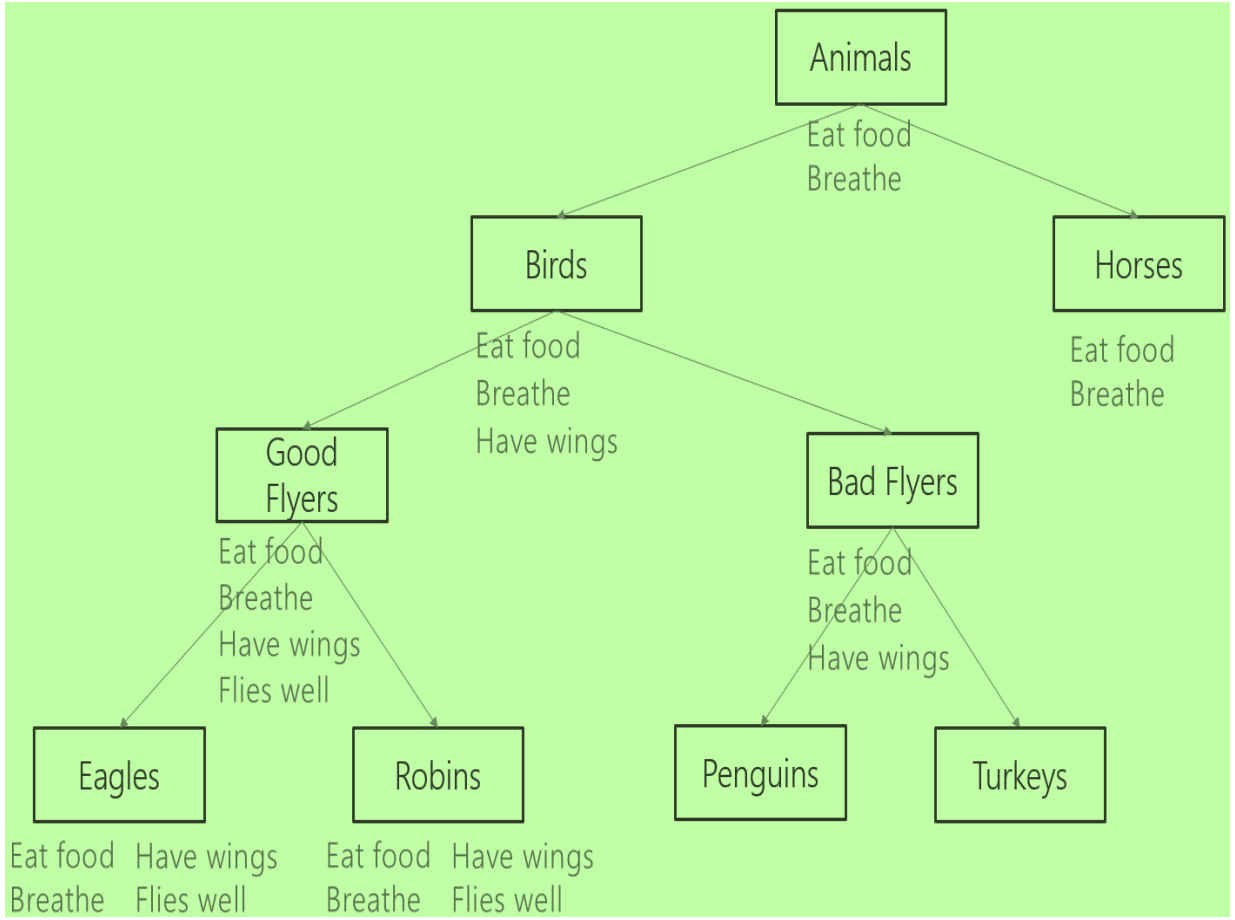
```
public class MyClass{
   private String field1;
   private static int field2;
}
```

  - Handy for:
    - Counting and statistics for all objects of the same class.
    - Shared values that all objects of the same class use (speed limit, gravity value, etc.)

To reference outside the class, like in the client class; we use class name, not object name.

```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();
int bikeCount = Bicycle.bikeCount;
```

Inheritance



- Formal Relations between classes'
  - SuperClasses and SubClasses
- SubClasses inherit behavior from the SuperClass
- SuperClasses are more generic, SubClasses are more specific.