Defining Your Own Class

James Brucker

Where's the Code?

In Java, all source code is contained in **classes**.

A **class** defines a *kind of object*.

Class defines the object's:

attributes, **behavior**, and **construction**.

You create objects from a class.

What is "static"?

Explained at end of slides

Class Structure

```
package coinpurse;
import java.util.List;
/**
 * Describe this class.
 * @author Your Name
 */
public class Coin {
    static attributes

    instance attributes

    constructors

    methods
}
```

This is the standard order of class members. Please use it.

Attributes

Attributes are what an object knows.

To refer to something, it must be a variable.

```
public class Coin {
    private double value;
    private String currency;



}
```

attributes of a Coin:

a Coin has a value and currency.

Not `static`

Declaring Attributes

```
public class Coin {
   /** value of coin */
   private double value;
```

Javadoc for attribute

Visibility

`public`

`protected`

`(default)`

`private`

Data Type

primitive

class name

interface

array

Variable Name

name of attribute

should start with lowercase

Initialize All Your Attributes!

```
public class Coin {
   private double value;
   private String currency = "THB";

   /** initialize a new coin */
   public Coin( double value ) {
      this.value = value ;
   }
```

Initialize attributes in either:

- assign a value as part of declaration, or
- (better) initialize in a constructor

Constructor Initializes a New Object

```
Coin ten = new Coin( 10 );
```

```
public class Coin {
    /** initialize a new coin */
    public Coin( double value ) {
         this.value = value ;
    }
```

Constructor has the same name as the class.

Does not have a return value. Not even "void".

this means "this object". "this.value" means the value attribute of *this* object.

`this` is used to resolve *ambiguity.*

How Objects are Created

```
c = new Coin( 10 )
```

JVM creates object in memory

initialize state of object by invoking a *constructor*

```
// constructor's job is to
// initialize a new object
public Coin( double value ) {
     this.value = value;
}
```

<u>Coin</u>
value=10.0
currency=THB

JVM returns the address of object

```
c = Coin@AE084D
```

What is wrong with this Code?

```
public class Coin {
   private double value;
   public void Coin(double value) {
       this.value = value;
   }
```

This code has legal syntax, but it is not a constructor.

More than One Constructor

```
public class Coin {
   public Coin( ) {
      this.value = 0;
      this.currency = "THB";
   }
   public Coin(double value) {
      this.value = value;
      this.currency = "THB";
   }
   public Coin(double value,
            String currency) {
      this.value = value;
      this.currency = currency;
   }
```

A class can have ***many*** *constructors*,

if they have different parameters.

Default Constructor

```
public class Coin {
  private double value;
  public Coin( ) {
     this.value = 0 ;
     this.currency = "THB";
  }
```

Coin zero = new Coin( );

A constructor with no parameters is called the default constructor.

Avoid Duplicate Code

```
public class Coin {
  public Coin( ) {
    this.value = 0;
    this.currency = "THB";
  }
  public Coin(double value) {
    this.value = value;
    this.currency = "THB";
  }
  public Coin(double value,String currency){
    this.value = value;
    this.currency = currency;
```

These 3 constructors all do the same thing.

Constructor calls Constructor

A constructor can call another constructor using "this()", but it must be the **first** statement in constructor.

```
public Coin( ) {
   this( 0.0, "THB");
}
public Coin(double value) {
   this( value, "THB");
}
public Coin(double value, String curr) {
    if (value < 0)
       throw new IllegalArgumentException(...);
    this.value = value;
    this.currency = curr;
}
```

Attributes for Knowing Things

An object has to remember information.

The attributes (defined in class) are what an object knows.

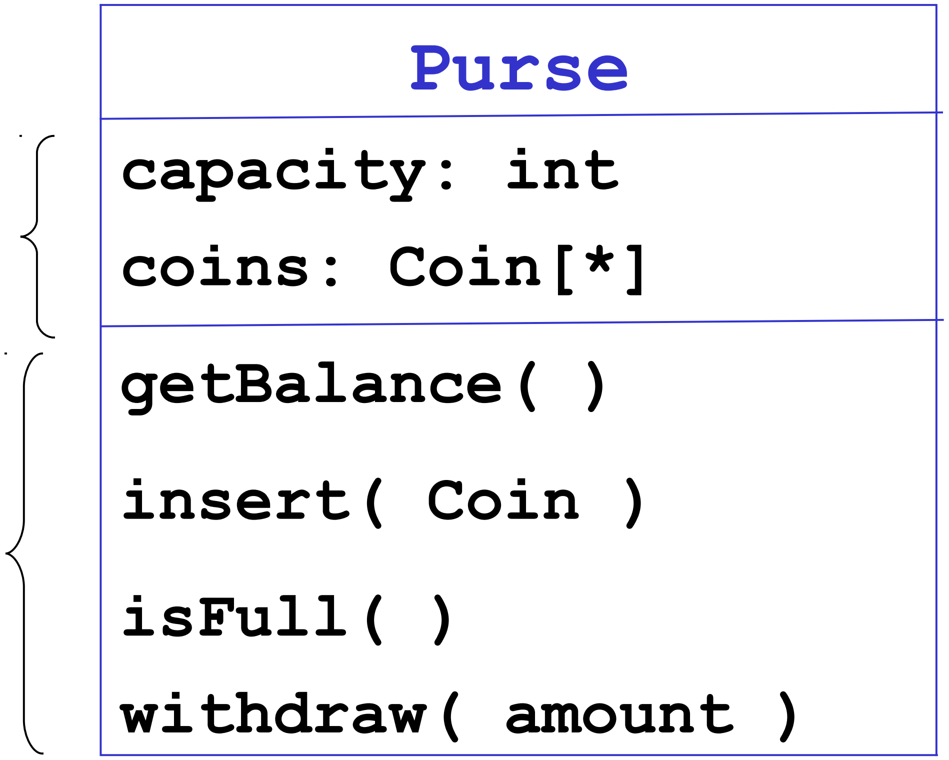
Attributes are what an object knows

Attributes -

what a Purse knows

Methods -

what a Purse can do



Defining an Attribute

Attributes should be defined near the start of class.

Attribute has a **visibility**, **data type**, and **name**.

You can optionally initialize its value.

Memory

0.0

```
class Coin {

      private double value = 0;
```

Accessibility:
`private`
`protected`
`(default)`
`public`

The type of value to store.

The name of this attribute

See the attributes of an Object

In BlueJ, you can "inspect" attributes of an object.

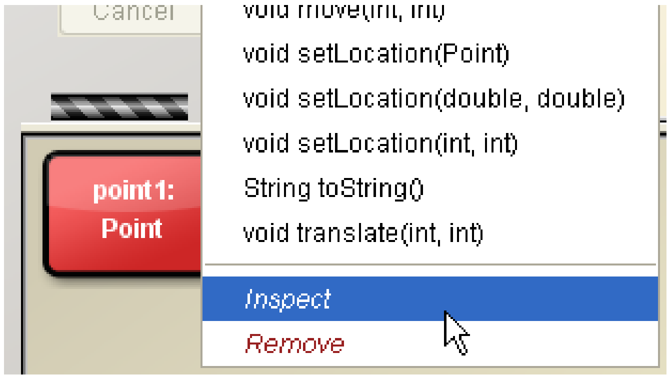
1. Create a new java.awt.Point:

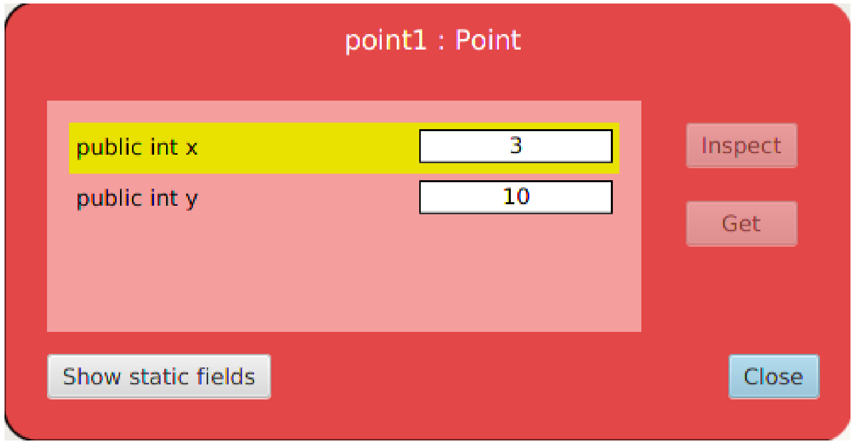
   Point p = new Point(3,10);

2. Right click and choose "*Inspect*".

3. What are the attributes?

Attributes of an object are also called "fields" or "properties".



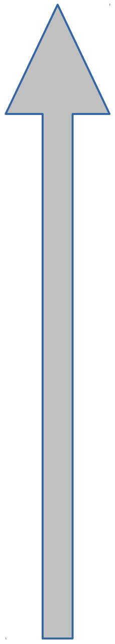


public int x 3
public int y 10

Visibility and Accessibility

The same rules apply to both attributes and methods.

most visible



public - can be accessed from any code, anywhere

protected - can only be accessed by this class, objects of this class, subclasses, or other classes/objects in same package

(**default**) - "package scope". Can be accessed by classes or objects in the same package as this class.

private - only this class and objects of this class can access

least visible

Private attributes

Private attribute can be accessed only by code in same class

```
public class Coin {
   private double value;
   public double add(Coin c) {
      return this.value + c.value;
   }
```

OK to access private attribute of another Coin

Cannot be accessed by other classes:

```
public class Purse {

   public double add(Coin c1, Coin c2) {

      return c1.value + c2.value;//ERROR
```

Protected attribute

Protected is mainly used for inheritance.
Protected also gives package-level access.

```
package coinpurse;
public class Coin {
    protected double value;
```

Can be accessed in other classes in same package:

```
package coinpurse;

public class Purse {

    public double add(Coin c1, Coin c2) {

        return c1.value + c2.value; //OK
```

Encapsulation

Protect your object's data from corruption!

Restrict access to object's attributes and methods.

attributes - usually **`private`**

methods - **`public`** for others to use

- **`private`** for "internal use only" code

- **`protected`** for use by subclasses and friends

Encapsulation Example

Coin hides its attributes, but provides "get" methods.

```
public class Coin {
   private double value;
   private String currency;

   public double getValue() {
        return value;
   }
   public String getCurrency() {
        return currency;
   }
```

Accessor methods: getValue()

An accessor method returns the value of an attribute.

Name begins with **get______( )**

Capitalize the next letter: getValue( ), getCurrency( )

```
   public double getValue() {
        return value;
   }
   public String getCurrency() {
        return currency;
   }
```

Boolean accessor: isOn(), hasX( )

Accessor method for boolean values begins with is___( ) or has___( ).

Capitalize the next letter: isOn( ), hasNext( )

```
public class LightBulb {
   /** Return true if light is on */
   public boolean isOn() {
        return on;
   }
```

Accessor can be a Computed Value

Some accessors compute the value on demand.

Example: GradeBook should **not** have a `total` attribute. Compute it as needed.

```
class GradeBook { // student scores
   private List<Double> scores;
   public double getTotal() {
      double total = 0.0;
      for(double s: scores) total += s;
      return total;
   }
   public void addScore(double score) {
       scores.add( score );
   }
```

`this` - always refers to "this object"

`this` is a special variable that refers to "this object".

Use "`this`" to resolve ambiguity in constructors and methods. But don't overuse it.

```
class Person {
   private String name;
   public Person(String name) {
      this.name = name;
   }
   public String getName() {
      return name;  // same as this.name
   }
```

this - sometimes used for clarity

equals() compares two people by name. We don't *really* need to write "this.name", but it is added for clarity.

```
class Person {
   private String name;

   /** Test if two people have same name */
   public boolean equals(Person other) {
      if (other==null) return false;
      return this.name.equals(
                           other.getName() );
   }
```

Note: you should not write `equals()` like this. Its done here for brevity.

3 Types of Variables

An object has access to 3 kinds of variables:

Attributes of the object

Static attributes of the class

Local variables and parameters (inside one method)

Local Variables

Variables defined inside a method are local variables.

(1) can only be used *inside the method*

(2) deleted when the method returns

```
public class Purse {

  public int getBalance( ) {
    int balance = 0;
    for(int k=0; k<coins.size(); k++) {
        // add coins.get(k) to balance
    }
```

Local variables are defined inside a method.

Local Variables vs. Attributes

An **attribute** is something an object *remembers* for its whole life.

A **local variable** is for temporary data. The value is lost when execution leaves the method.

A purse must *remember* its capacity and coins

```
public class Purse {
  private int capacity;
  private List coins;
  public int getBalance( ) {
    int balance = ...;
    return balance;
  }
```

balance is computed each time we need it.

Don't need to remember.

Person refers to Person

An object can have attributes that **refer** to other objects of the same class.   This is quite common.

```
class Person {
   private String name;
   private Person father;
   private Person mother;
   public Person(String name) {
      this.name = name;
   }
   public void setFather(Person f) {
      father = f;
   }
```

Methods

- ✓ The behavior of objects is defined in methods.
- ✓ Methods contain the program's logic.

name of method

```
String makeMessage(int guess, int secret ) {
        if guess == secret
                return "You're right!"
        else if guess < secret
                return "guess is too small"
        else return "guess is too large"
}
```

instructions for this method

static: class attributes & methods

static members (attributes and methods) are provided by the class, but...

Not associated with any object.

```
// static method of String class
String.format("total is %2.f", total);
// instance method - associated with a
// String object
String s = "hello, nerd";
int n = s.length();
```

Objects can access static members

Student object can access static **`nextId`** field.

```java
public class Student {
    static long nextId = 6010540001L;
    private long id; // id of this student
    private String name;
    /** initialize a new student */
    public Student(String name) {
        this.name = name;
        this.id = nextId;
        nextId++;
    }
```

Static methods cannot access instance members

Static code cannot access object attributes or methods

```
public class Person {
    private String name;
    public String toString() {
        return "My name is "+name;
    }

    public static void main(String[] args){
        System.out.println( name ); ERROR
        System.out.println(
                Person.toString() ); ERROR
```

Static Method as Service

Static methods are often "services". Something that the class does, but is not associated with any object.

Get the current system time in milliseconds:

```
System.currentTimeMillis(  );
```

Name of Class

static method name

Utility methods provided by class

Square root:

```
    double r = Math.sqrt( 2 );
```

Get the int value of a String:

```
    int value = Integer.parseInt("123");
```

Get the name of current user (a service):

```
  String who = System.getProperty("user.name");
```

These methods are performed by the class, not an object.

Writing static methods

You already know this.

```
/** distance between points (x1,y1) and (x2,y2) */
public static double distance( x1, y1, x2, y2 ) {
   // hypot computes hypothenous of a triangle
   double d = Math.hypot( x1 - x2, y1 - y2 );
   return d;
}

public static void main(String[] args) {
   // start the application
}
```

Find the Errors

2 syntax errors and 1 semantic error (but syntax is legal).

```
public class Person {
    private static String name;
    /** initialize a new person */
    public Person(String name) {
        this.name = name;
    }
    public void setName(String newname){
        name = newname;
    }
    public static void main(String[] args) {
        this.setName( args[0] );
    }
```

Find the Error

What is wrong with this code? How to correct it?

It returns correct value only the first time it is called.

```
public class GradeBook {
   private double[] scores = ...;
   private double total = 0.0;

   public double getTotal( ) {
      for( double score: scores ) {
         total += score;
      }
      return total;
   }
```