

More With Classes, Fields & Constructors

Brandon Krakowsky



Penn
Engineering

Classes

Classes - Review

- Everything in Java is object-oriented and class-based
 - This means you have to create *at least one class* to write a Java program



Classes - Review

- Everything in Java is object-oriented and class-based
 - This means you have to create *at least one class* to write a Java program
- A class describes an object
 - It's like a *template* for a new kind of object
 - When you define a class, you're defining a new *data type*



Classes - Review

- Everything in Java is object-oriented and class-based
 - This means you have to create *at least one class* to write a Java program
- A class describes an object
 - It's like a *template* for a new kind of object
 - When you define a class, you're defining a new *data type*
- To use the *object*, you create an *instance* of the class



Classes - Review

- Everything in Java is object-oriented and class-based
 - This means you have to create *at least one class* to write a Java program
- A class describes an object
 - It's like a *template* for a new kind of object
 - When you define a class, you're defining a new *data type*
- To use the *object*, you create an *instance* of the class
- A class includes:
 - Fields (instance variables) that hold the data for each object
 - Constructors that describe how to create a new object instance of the class
 - Methods that describe the actions the object can perform



Classes - Review

- Everything in Java is object-oriented and class-based
 - This means you have to create *at least one class* to write a Java program
- A class describes an object
 - It's like a *template* for a new kind of object
 - When you define a class, you're defining a new *data type*
- To use the *object*, you create an *instance* of the class
- A class includes:
 - Fields (instance variables) that hold the data for each object
 - Constructors that describe how to create a new object instance of the class
 - Methods that describe the actions the object can perform

We'll look at Fields and Constructors today!



Defining a Class - Review

- Here's simple syntax for defining a sample class:

```
public class ClassName {  
    // The fields (instance variables) of the object  
    // The constructors for creating the object  
    // The methods for communicating with the object  
}
```



Defining a Class - Review

- Here's simple syntax for defining a sample class:

```
public class ClassName {  
    // The fields (instance variables) of the object  
    // The constructors for creating the object  
    // The methods for communicating with the object  
}
```

- `public` is an *access modifier* that defines the visibility of the class
 - `public` means any other program in the Java project can use the class (i.e., create instances or call methods)
 - We'll talk about other *access modifiers* later in the course

Defining a Class - Review

- Here's simple syntax for defining a sample class:

```
public class ClassName {  
    // The fields (instance variables) of the object  
    // The constructors for creating the object  
    // The methods for communicating with the object  
}
```

- `public` is an *access modifier* that defines the visibility of the class
 - `public` means any other program in the Java project can use the class (i.e., create instances or call methods)
 - We'll talk about other *access modifiers* later in the course
- Things in a class can be in any order



Fields

Defining Fields in a Class

- An object's data is stored in fields (instance variables)
 - The fields describe the state of the object
 - Fields are defined as variable declarations in the class
- Sample class definition with instance variables:

```
public class ClassName {  
    // The fields (instance variables) of the object  
    String name; //declaration to store a String in the object,  
    defaults to null  
    double health; //declaration to store a double in the object  
    int age = 0; //declaration to store an int in the object, initially  
    set to 0  
}
```



Defining Fields in a Class

- An object's data is stored in fields (instance variables)
 - The fields describe the state of the object
 - Fields are defined as variable declarations in the class
- Sample class definition with instance variables:

```
public class ClassName {  
    // The fields (instance variables) of the object  
    String name; //declaration to store a String in the object,  
defaults to null  
    double health; //declaration to store a double in the object  
    int age = 0; //declaration to store an int in the object, initially  
set to 0  
}
```

- Fields are available throughout the entire class that declares them



Constructors

Defining a Constructor for a Class

- A constructor is code to create an object
- The syntax for a constructor is **very similar** to a method:

```
public ClassName(parameters) {  
    //code using parameters to set up initial state of object  
}
```



Defining a Constructor for a Class

- A constructor is code to create an object
- The syntax for a constructor is **very similar** to a method:

```
public ClassName(parameters) {  
    //code using parameters to set up initial state of object  
}
```

- public means the constructor is accessible by any other program in the Java project



Defining a Constructor for a Class

- A constructor is code to create an object
- The syntax for a constructor is **very similar** to a method:

```
public ClassName(parameters) {  
    //code using parameters to set up initial state of object  
}
```

- public means the constructor is accessible by any other program in the Java project
- ClassName has to be the same name as the class that the constructor occurs in



Defining a Constructor for a Class

- A constructor is code to create an object
 - The syntax for a constructor is **very similar** to a method:

```
public ClassName(parameters) {  
    //code using parameters to set up initial state of object  
}
```

- `public` means the constructor is accessible by any other program in the Java project
 - `ClassName` has to be the same name as the class that the constructor occurs in
 - The constructor `parameters` are a comma-separated list of variable declarations

Defining a Constructor for a Class

- Sample class definition with constructor:

```
public class ClassName {  
    // The fields (instance variables) of the object  
    String name; //declaration to store a String in the object,  
defaults to null  
    double health; //declaration to store a double in the object  
    int age = 0; //declaration to store an int in the object, initially  
set to 0  
  
    // The constructor for creating the object  
    public ClassName(parameters) {  
        //code using parameters to set up initial state of object  
    }  
}
```

Dog Program

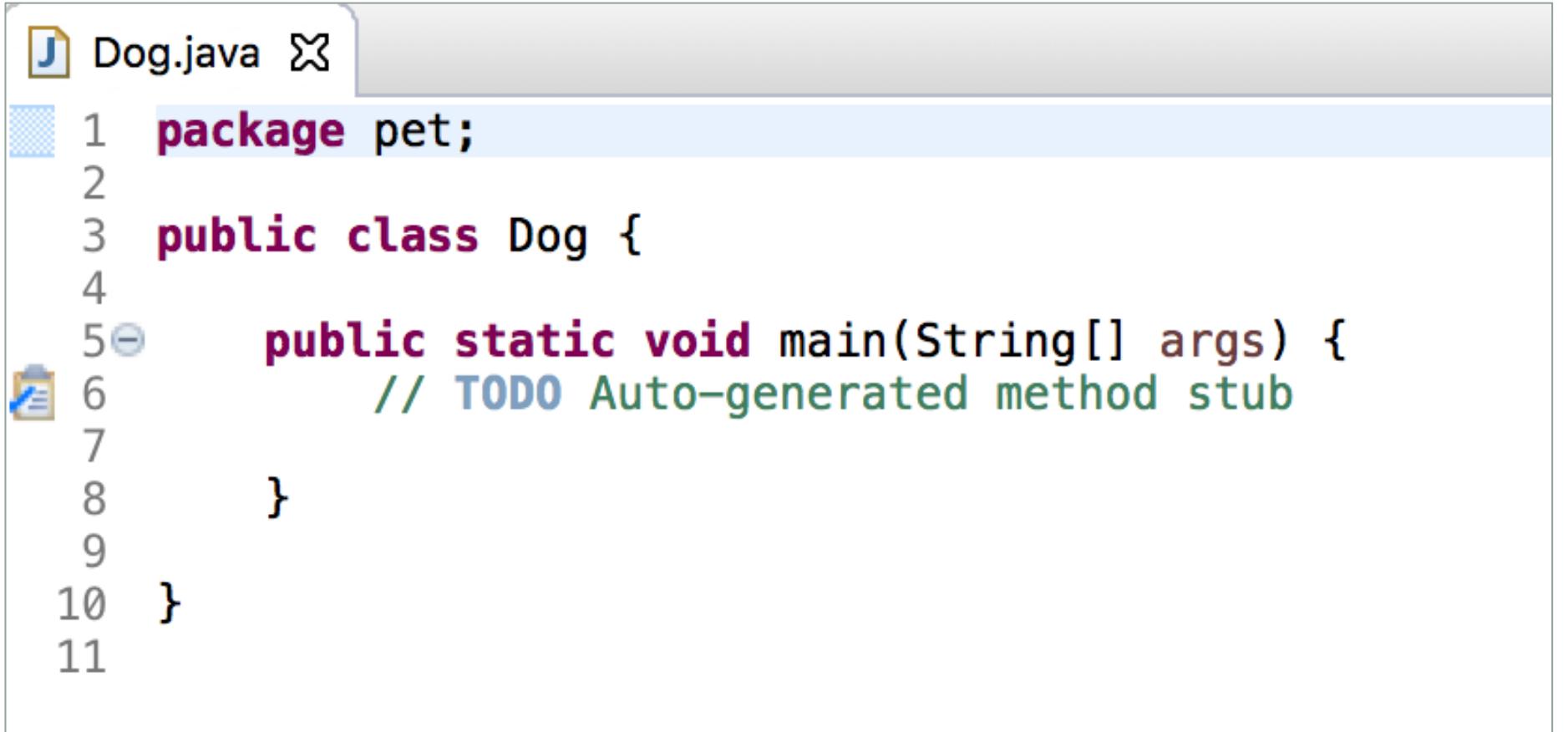
Dog Program

- In Eclipse, create a new project
- Create 2 classes:
 - Dog
 - Provide the package name “pet”
 - Make sure public static void main(String[] args) IS checked
 - Owner
 - Provide the package name “pet”
 - Make sure public static void main(String[] args) IS NOT checked



Dog Class

- Create a new *Dog* class in the *pet* package
 - This class will represent a Dog



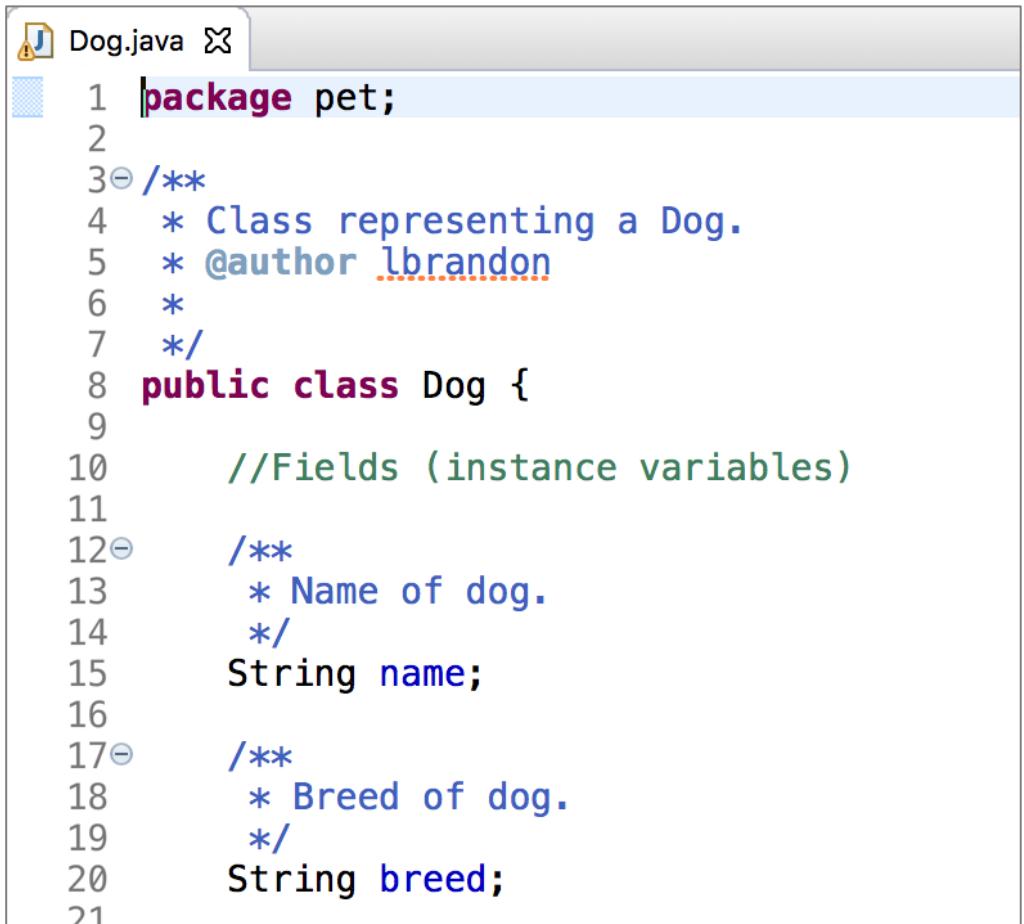
The image shows a screenshot of a Java code editor with a file named "Dog.java". The code is as follows:

```
1 package pet;
2
3 public class Dog {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8     }
9
10 }
```

The code editor interface includes a toolbar at the top with icons for file operations like New, Open, Save, and Close. The code is syntax-highlighted with purple for keywords like package, public, class, and static, and green for comments. Line numbers are visible on the left side of the code area.

Dog Class – Instance Variables

- Make sure to add javadocs to your classes, variables, and methods as you write your program
- Create some attributes (instance variables) for your Dog



The image shows a screenshot of a Java code editor with a file named "Dog.java". The code defines a class "Dog" with two instance variables: "name" and "breed". Each variable has a corresponding Javadoc comment describing its purpose.

```
1 package pet;
2
3 /**
4 * Class representing a Dog.
5 * @author lbrandon
6 *
7 */
8 public class Dog {
9
10    //Fields (instance variables)
11
12    /**
13     * Name of dog.
14     */
15    String name;
16
17    /**
18     * Breed of dog.
19     */
20    String breed;
21}
```



Dog Class – Instance Variables

- Make sure to add javadocs to your classes, variables, and methods as you write your program
- Create some attributes (instance variables) for your Dog
 - The *Owner* data type doesn't exist yet. We'll create it as another class soon.

```
21
22    /**
23     * Gender of dog.
24     */
25    char gender;
26
27    /**
28     * Age of dog.
29     */
30    int age;
31
32    /**
33     * Weight of dog.
34     */
35    double weight;
36
37    /**
38     * Dog's owner.
39     */
40    Owner owner;
41
```

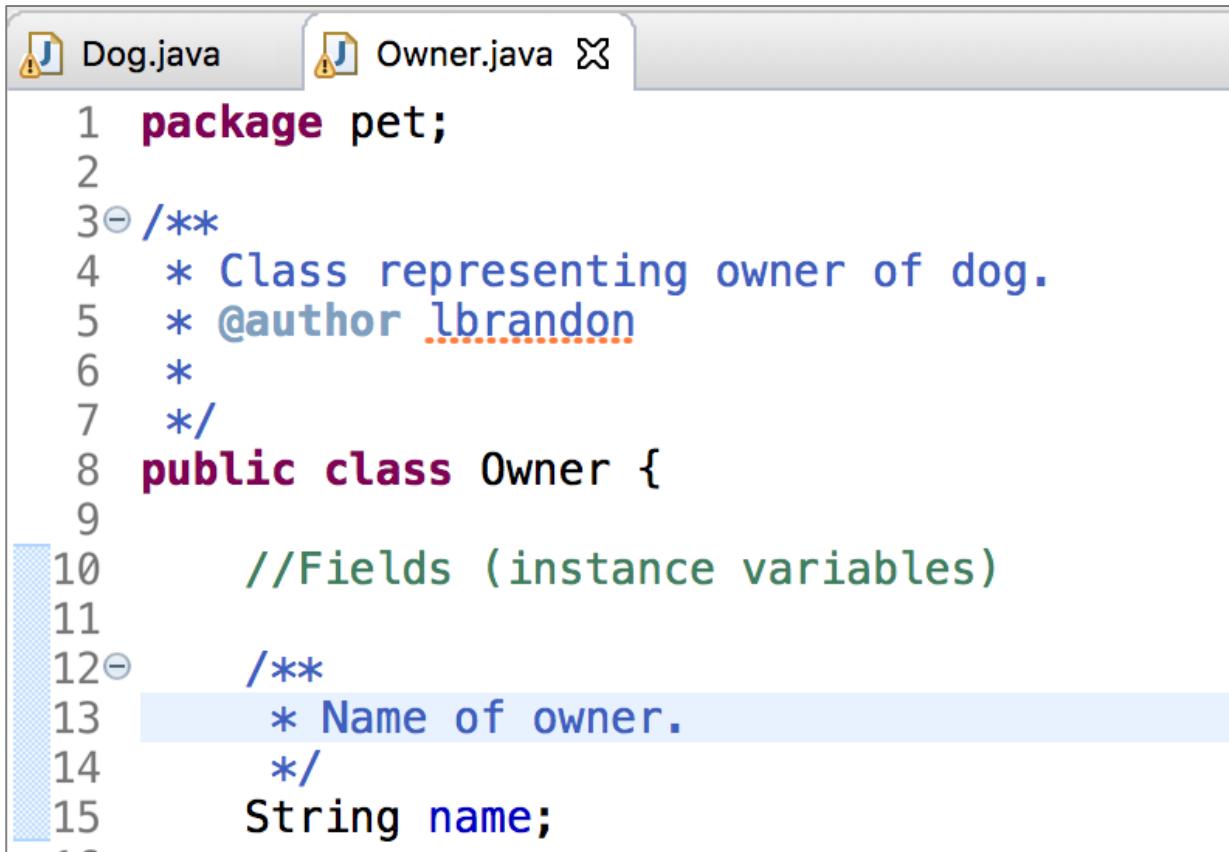
Dog Class - Constructor

- Create a constructor for your object (class)
 - Again, the *Owner* data type doesn't exist yet.

```
41
42     //constructor
43
44④    /**
45     * Constructor to create a dog with given name, breed, gender, and owner.
46     * @param name of dog
47     * @param breed of dog
48     * @param gender of dog ('m' or 'f')
49     * @param owner of dog
50     */
51④    public Dog(String name, String breed, char gender, Owner owner) {
52
53        //setting values for instance of Dog from arguments
54        this.name = name;
55        this.breed = breed;
56        this.gender = gender;
57        this.owner = owner;
58
59        //setting default values for instance of Dog
60        this.age = 1;
61        this.weight = 10;
62    }
63
```

Owner Class – Instance Variables

- Create a new *Owner* class in the *pet* package
 - This will represent a Dog owner
- Create some attributes (instance variables) for the Dog owner



```
1 package pet;
2
3 /**
4  * Class representing owner of dog.
5  * @author lbrandon
6  *
7 */
8 public class Owner {
9
10    //Fields (instance variables)
11
12 /**
13  * Name of owner.
14  */
15 String name;
```



Owner Class – Constructor & Methods

- Create a constructor for your object (class) and other methods

```
10
17      //constructor
18  /**
19   * Creates a dog owner.
20   * @param name of owner
21   */
22  public Owner (String name) {
23      this.name = name;
24  }
25
26  //methods
27
28  /**
29   * Hears dog bark and responds.
30   */
31  public void receiveBark() {
32      //prints receiving bark String
33      System.out.println("ok got it, take it easy ...");
34  }
35
36  /**
37   * Gets name of owner.
38   * @return owner's name
39   */
40  public String getName() {
41      return this.name;
42  }
43
```



Dog Class - Methods

- Create an *eat* method in the *Dog* class

```
63
64     //methods
65
66     /**
67      * Tells dog to eat given amount of given food.
68      * @param food to eat
69      * @param amount of food
70     */
71     public void eat(String food, double amount) {
72         //print useful eating String
73         System.out.println(this.name + " is eating " + food);
74
75         //calculate weight gain
76         double weightGained = 0.01 * amount;
77
78         //add weight gain
79         this.weight += weightGained;
80     }
81
```

Dog Class - Methods

- Create a *haveBirthday* method in the *Dog* class

```
//  
78     /**  
79      * Dog has a birthday and increments age by 1.  
80      * @return new age  
81     */  
82     public int haveBirthday() {  
83         //print useful birthday String  
84         System.out.println(this.name + " is having a birthday");  
85  
86         //increment age  
87         this.age++;  
88  
89         return this.age;  
90     }  
91 }
```

Dog Class - Methods

- Create a *bark* method in the *Dog* class

```
91
92        /**
93     * Dog barks at owner.
94     */
95        public void bark() {
96        //print bark String
97        System.out.println("bark bark bark ..!!!");
98
99        //call method in Owner class
100       this.owner.receiveBark();
101
102 }
```

Dog Class - Methods

- Create a *getDogInfo* method in the *Dog* class

```
102
103     /**
104      * Gets dog's basic info.
105      * @return String with info
106     */
107     public String getDogInfo() {
108         return this.name + " " + this.breed + " " + this.gender;
109     }
110
```

Dog Class - Methods

- Other methods in the *Dog* class

```
14
15    /**
16     * Get dog's age
17     * @return dog's age
18     */
19    public int getAge() {
20        return this.age;
21    }
22
23    /**
24     * Get dog's weight
25     * @return dog's weight
26     */
27    public double getWeight() {
28        return this.weight;
29    }
30
31    /**
32     * Get dog's owner's info
33     * @return Owner's name
34     */
35    public String getOwnerInfo() {
36        return this.owner.getName();
37    }
38
```



Dog Class – *main* Method

- Create instances of the *Owner* and *Dog* classes in your *main* method

```
134  
135     public static void main(String[] args) {  
136  
137         //creating instance of Owner class with name  
138         Owner brandon = new Owner("Brandon");  
139  
140         //creating instance of Dog class with name, breed, gender, and owner  
141         Dog princess = new Dog("Princess", "Shih Tzu", 'f', brandon);  
142  
143         //get and print Princess' info  
144         System.out.println(princess.getDogInfo());  
145  
146         //get and print Princess' owner's info  
147         System.out.println("princess' owner: " + princess.getOwnerInfo());  
148
```

Dog Class – *main* Method

```
145  
146     //get and print Princess' weight  
147     System.out.println("weight before eating: " + princess.getWeight());  
148  
149     princess.eat("dog food", 30);  
150  
151     //get and print Princess' weight  
152     System.out.println("weight after eating: " + princess.getWeight());  
153  
154     princess.eat("dog food", 30);  
155  
156     //get and print Princess' weight  
157     System.out.println("weight after eating: " + princess.getWeight());  
158
```

Dog Class – *main* Method

```
58  
59     //get and print Princess' age  
60     System.out.println("age before birthday: " + princess.getAge());  
61  
62     int newAge = princess.haveBirthday();  
63  
64     //get and print Princess' age  
65     System.out.println("age after birthday: " + newAge);  
66  
67     newAge = princess.haveBirthday();  
68  
69     //get and print Princess' age  
70     System.out.println("age after birthday: " + newAge);  
71
```

Dog Class – *main* Method

```
174  
175     //princess barks at owner  
176     princess.bark();  
177  
178     //create another instance of Dog class (another dog)  
179     Dog fido = new Dog("Fido", "Pug", 'm', new Owner("John"));  
180  
181     //get fido's info  
182     System.out.println(fido.getDogInfo());  
183  
184     //get fido's owner's info  
185     System.out.println(fido.getOwnerInfo());  
186  
187     //fido's weight  
188     System.out.println("fido's weight before eating: "+ fido.getWeight());  
189  
190     fido.eat("lettuce", 2);  
191  
192     //fido's weight  
193     System.out.println("fido's weight after eating: " + fido.getWeight());  
194
```

Banking Program

Banking Program

- In Eclipse, create a new “Banking” project
- Create 3 classes:
 - Bank
 - Provide the package name “banking”
 - Make sure `public static void main(String[] args)` IS checked
 - BankAccount
 - Provide the package name “banking”
 - Make sure `public static void main(String[] args)` IS NOT checked
 - Customer
 - Provide the package name “banking”
 - Make sure `public static void main(String[] args)` IS NOT checked

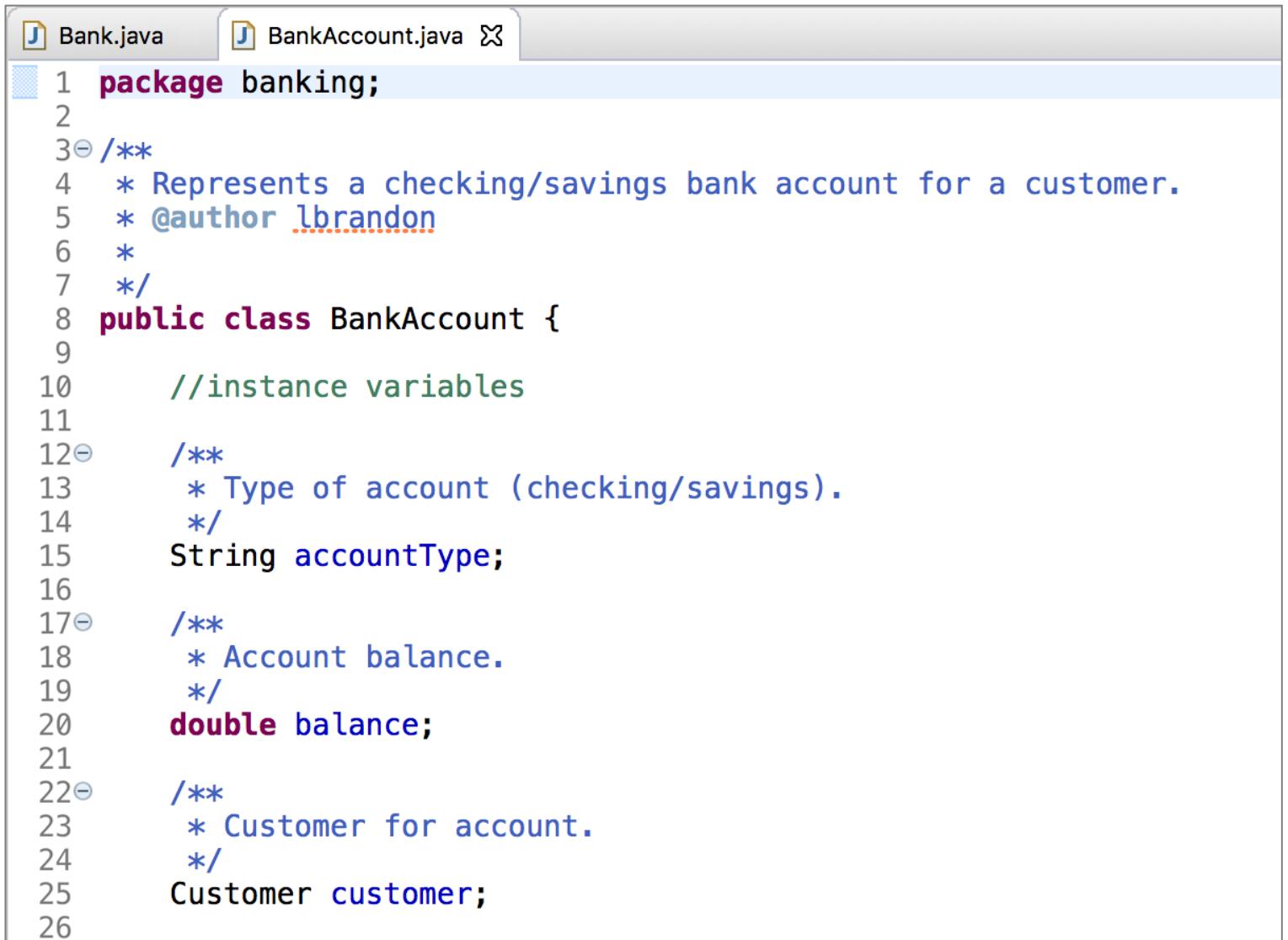


Bank Class

```
J Bank.java ✎
1 package banking;
2
3 import java.util.Scanner;
4
5 /**
6  * Represents a bank for managing customers and their bank accounts.
7  * @author lbrandon
8  *
9 */
10 public class Bank {
11
12     /**
13      * Create new Bank and runs the program.
14      * @param args
15      */
16     public static void main(String[] args) {
17         //creates new instance of Bank class
18         Bank bank = new Bank();
19
20         //calls the run method in Bank class
21         bank.run();
22     }
23
24     /**
25      * Runs the program by initializing and managing, bank accounts and customers.
26      */
27     public void run() {
28 }
```



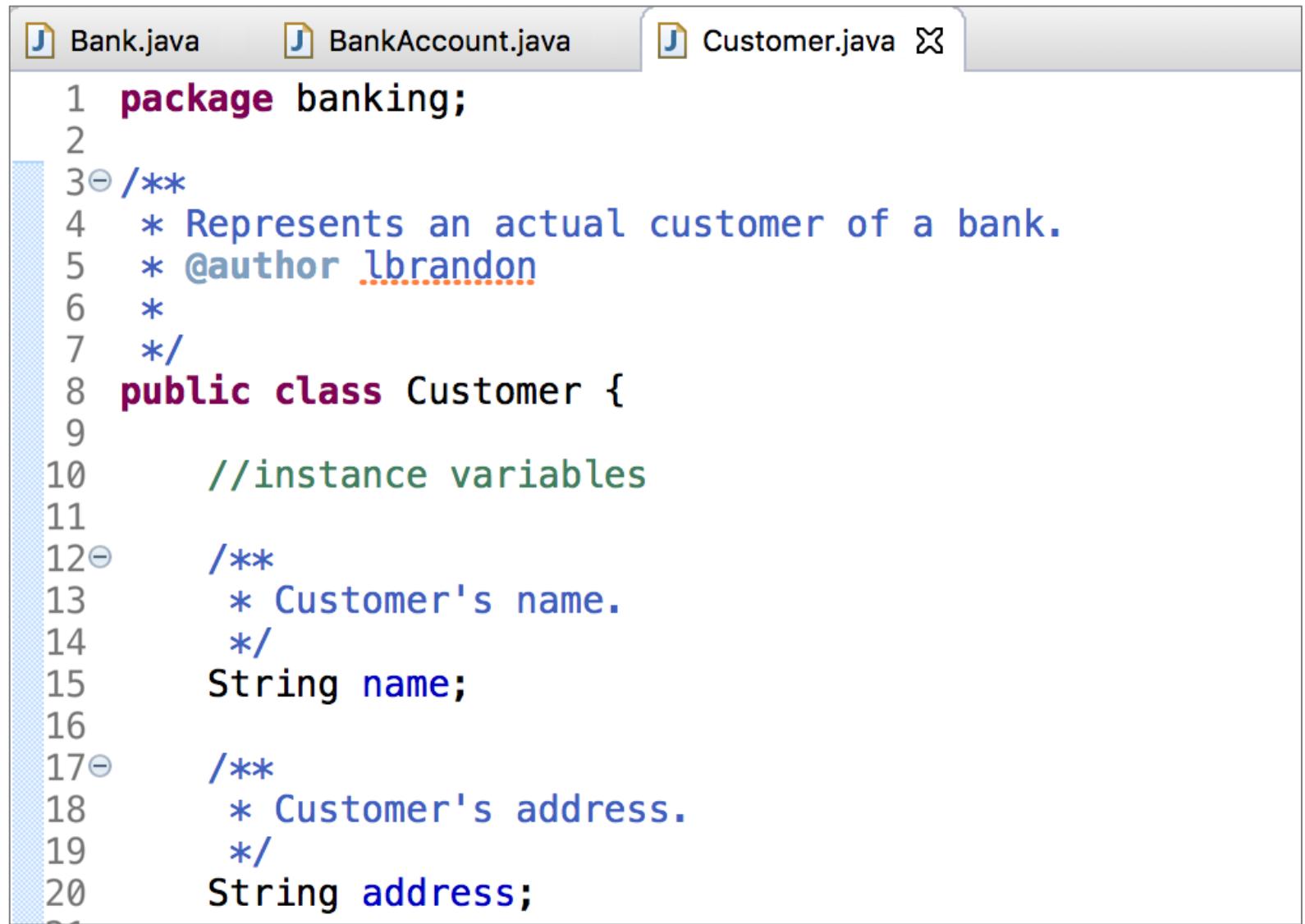
BankAccount Class



The image shows a screenshot of a Java code editor with two tabs: "Bank.java" and "BankAccount.java". The "BankAccount.java" tab is active, displaying the following code:

```
1 package banking;
2
3 /**
4  * Represents a checking/savings bank account for a customer.
5  * @author lbrandon
6  *
7 */
8 public class BankAccount {
9
10    //instance variables
11
12    /**
13     * Type of account (checking/savings).
14     */
15    String accountType;
16
17    /**
18     * Account balance.
19     */
20    double balance;
21
22    /**
23     * Customer for account.
24     */
25    Customer customer;
26}
```

Customer Class



The screenshot shows a Java code editor with three tabs: Bank.java, BankAccount.java, and Customer.java. The Customer.java tab is active, indicated by a blue selection bar and a larger font size. The code itself is as follows:

```
1 package banking;
2
3 /**
4  * Represents an actual customer of a bank.
5  * @author lbrandon
6  *
7 */
8 public class Customer {
9
10    //instance variables
11
12    /**
13     * Customer's name.
14     */
15    String name;
16
17    /**
18     * Customer's address.
19     */
20    String address;
```



Customer Class

```
--  
22     //constructor  
23  
24     /**  
25      * Creates a customer with the given name.  
26      * @param name of customer  
27      */  
28     public Customer(String name) {  
29         this.name = name;  
30     }  
31
```

Customer Class

```
31
32     //methods
33
34-    /**
35     * Sets address for customer.
36     * @param address for customer
37     */
38-    public void setAddress(String address) {
39         this.address = address;
40     }
41
42-    /**
43     * Returns customer's name.
44     * @return name of customer
45     */
46-    public String getName() {
47         return this.name;
48     }
49
50-    /**
51     * Returns customer's address.
52     * @return address of customer
53     */
54-    public String getAddress() {
55         return this.address;
56     }
```



BankAccount Class

```
27  
28     //constructor  
29  
30     /**  
31      * Creates a bank account of given type for given customer.  
32      * @param accountType for bank account  
33      * @param customer for this account  
34      */  
35     public BankAccount(String accountType, Customer customer) {  
36         this.accountType = accountType;  
37         this.customer = customer;  
38     }  
39
```

BankAccount Class

```
59  
40     //methods  
41  
42     /**  
43     * Deposits the given balance.  
44     * @param balance to add  
45     */  
46     public void deposit(double balance) {  
47        this.balance += balance;  
48    }  
49  
50     /**  
51     * Withdraws the given amount.  
52     * @param amount to subtract  
53     * @throws Exception if amount is greater than available balance  
54     */  
55     public void withdraw(double amount) throws Exception {  
56        if (amount > this.balance) {  
57            throw new Exception("Amount is greater than available balance.");  
58        }  
59        this.balance -= amount;  
60    }  
61
```



BankAccount Class

```
64
65    /**
66     * Gets account type and balance.
67     * @return String with all info
68     */
69    public String getAccountInfo() {
70        return this.accountType + ": " + this.getBalance();
71    }
72
73    /**
74     * Gets rounded account balance.
75     * @return balance rounded to 2 decimal places
76     */
77    public double getBalance() {
78        BigDecimal bd = new BigDecimal(Double.toString(this.balance));
79        bd = bd.setScale(2, RoundingMode.HALF_UP);
80
81        return bd.doubleValue();
82    }
83
84    /**
85     * Gets customer name and address.
86     * @return String with all info
87     */
88    public String getCustomerInfo() {
89        return this.customer.getName() + " from " + this.customer.getAddress();
90    }
91
```

Bank Class

```
23  
24   */  
25  * Runs the program by initializing and managing, bank accounts and customers.  
26  */  
27  public void run() {  
28  
29      //scanner for user input  
30      Scanner scanner = new Scanner(System.in);  
31  
32      System.out.println("Welcome to the Bank! What's your name? ");  
33  
34      //get name  
35      String name = scanner.next();  
36  
37      System.out.println("Hello " + name + "! We're creating checking and savings accounts for you!");  
38  
39      //create customer  
40      Customer customer = new Customer(name);  
41  
42      //get address|  
43      System.out.println("What's your address? ");  
44      String address = scanner.next();  
45      customer.setAddress(address);  
46
```

Bank Class

```
46  
47     //create checking account for customer  
48     //calls BankAccount constructor  
49     BankAccount checkingAccount = new BankAccount("checking", customer);  
50  
51     //create savings account for same customer  
52     //calls BankAccount constructor  
53     BankAccount savingsAccount = new BankAccount("savings", customer);  
54  
55     //get customer info for checking account  
56     System.out.println("For customer: " + checkingAccount.getCustomerInfo());  
57  
58     //get account info for checking account  
59     System.out.println(checkingAccount.getAccountInfo());  
60  
61     //get account info for savings account  
62     System.out.println(savingsAccount.getAccountInfo());  
63
```

Bank Class

```
65  
64    //deposits  
65  
66    //into checking  
67    System.out.println();  
68    System.out.println("Amount (decimal) to deposit into your checking? ");  
69    double amount = scanner.nextDouble();  
70    checkingAccount.deposit(amount);  
71  
72    //into savings  
73    System.out.println("Amount (decimal) to deposit into your savings? ");  
74    amount = scanner.nextDouble();  
75    savingsAccount.deposit(amount);  
76  
77    //print current balances  
78    System.out.println("Checking balance: " + checkingAccount.getBalance());  
79    System.out.println("Savings balance: " + savingsAccount.getBalance());  
80
```

Bank Class

```
81     //withdrawals
82     //from checking
83     System.out.println();
84     System.out.println("Amount (decimal) to withdraw from your checking? ");
85     amount = scanner.nextDouble();

86
87     //Java forces us to "catch" the possible exception (error)
88     try {
89         checkingAccount.withdraw(amount);
90     } catch (Exception e) {
91         //prints the message associated with the custom Exception
92         //additionally, prints the stack trace (code leading to error)
93         //e.printStackTrace();
94
95         //retrieves and prints just the message associated with the custom Exception
96         System.out.println(e.getMessage());
97     }
98 }
```

Bank Class

```
98
99     System.out.println("Enter amount to withdraw from savings: ");
100    amount = scanner.nextDouble();
101    try {
102        savingsAccount.withdraw(amount);
103    } catch (Exception e) {
104        //prints the message associated with the custom Exception
105        //additionally, prints the stack trace (code leading to error)
106        //e.printStackTrace();
107
108        //retrieves and prints just the message associated with the custom Exception
109        System.out.println(e.getMessage());
110    }
111
112    //print current balances
113    System.out.println("Final checking balance: " + checkingAccount.getBalance());
114    System.out.println("Final savings balance: " + savingsAccount.getBalance());
115
116    //always close scanner
117    scanner.close();
118 }
```

Homework 5

Homework 5

Will be assigned by Tuesday, October 4th and due Thursday, October 13th at midnight

- In this HW, you will be implementing a game called **Pick Your Letters**, which is a game that involves re-arranging a group of letters in order to make up a word
- This game needs two players -- we will just play the user versus the computer. The user's moves are decided by the user playing the game, by asking for input, and the computer's moves are decided by the program.
- The topics are:
 - Arrays
 - Strings
 - Class-based object-oriented programming
- To complete the assignment:
 - Submit all the classes in your entire Java project. This will include everything in your “src” folder.

