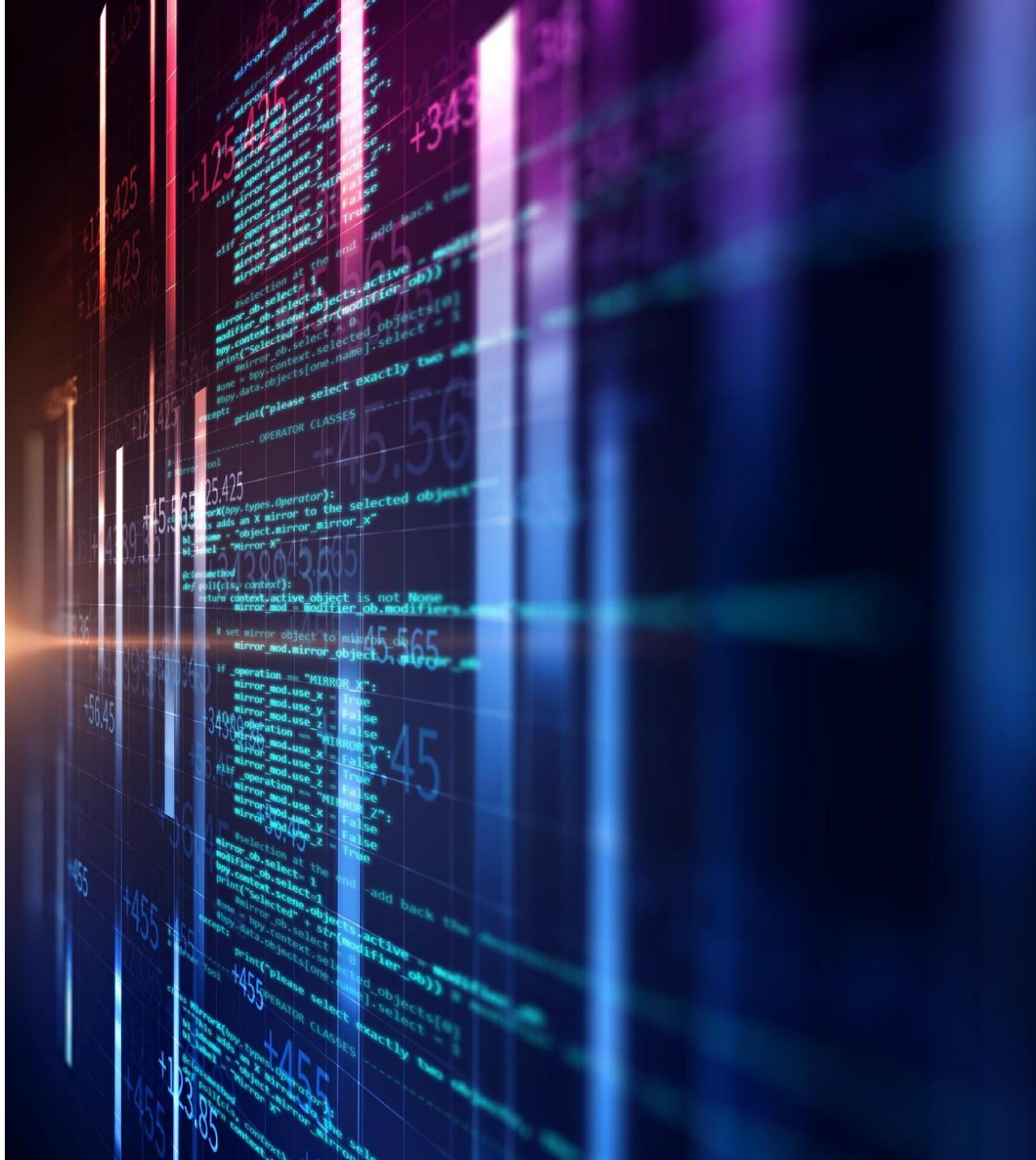
# Classes

Brandon Krakowsky

Penn Engineering



# Classes

# Classes

- 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
  - It's a very similar concept in Python
- 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

# Defining a Class

- 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

# 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

# Defining a Constructor for a Class

- A constructor is code to create an object
- The syntax for a constructor is:

```
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
    }
}
```

# Defining a Method in a Class

- A method is a function in an object that allows you to use and communicate with that object
- The syntax for a method is:

```
return-type methodName(parameters) {
    // locally defined variables
    // code using parameters
}
```

- If a method is to return a result, return-type is the data type of the result
  - You must use a return statement to exit the method with a result of the correct type
- If a method doesn't return a result, return-type is void
  - This indicates that a method doesn't return a value
  - In this case, you don't need to use a return statement to exit the method

# Defining a Method in a Class

- Sample class definition with a method:

```
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
    }

    // A method for communicating with the object
    String getName(parameters) {
        //returns value of “name” instance variable
        //”this” refers to this instance of the class (ClassName)
        return this.name;
    }
}
```

# Creating an Instance of a Class

- To use a class, you create an instance of the object by calling its constructor and using the keyword new
- Here's syntax to define a class and to create an instance:

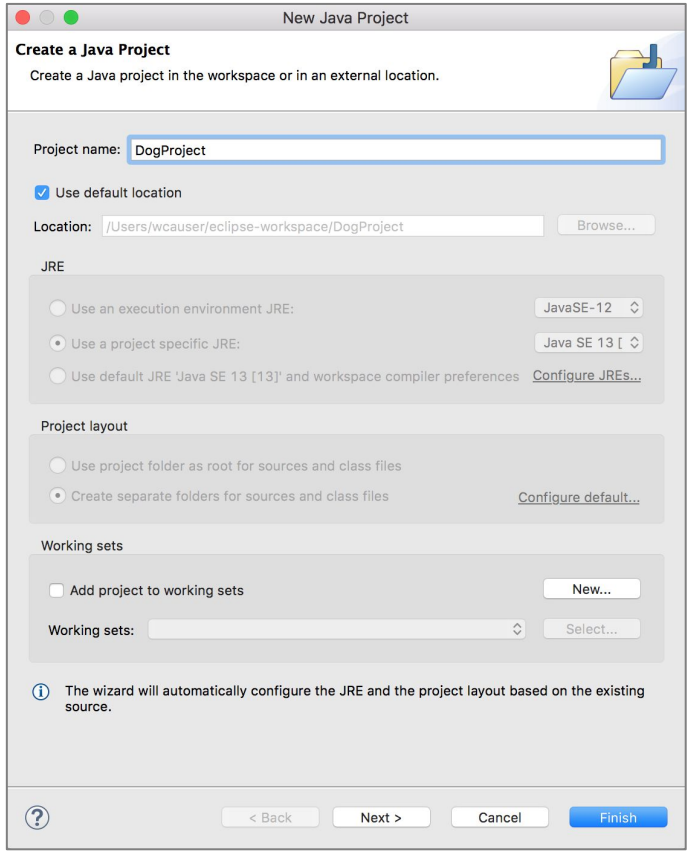
```
public class ClassName {

    public ClassName(par1, … parN) {
        //code using parameters to set up initial state of object
    }

    public static void main(String[] args) {
        //create instance of ClassName
        ClassName c = new ClassName(arg1, .., argN);
    }
}
```

- new creates a new instance of the object

# Dog Project

# Dog Project

- In Eclipse, go to “File” → “New” → “Project”




Provide a Project name
- Project names should be capitalized

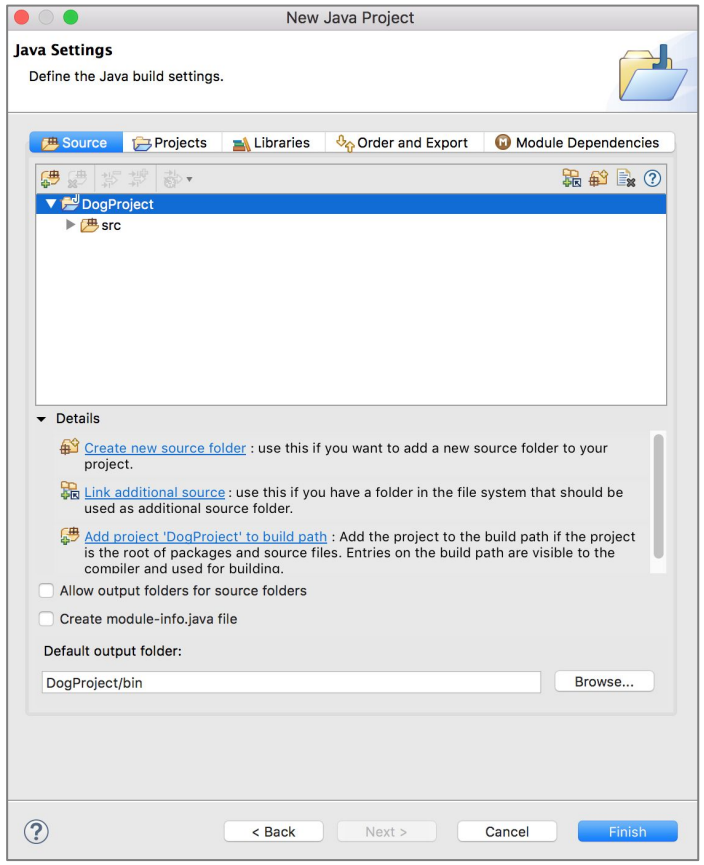
Use the default location

Use the default JRE and project layout

Click “Next”

# Dog Project

- Define the compilation/build settings



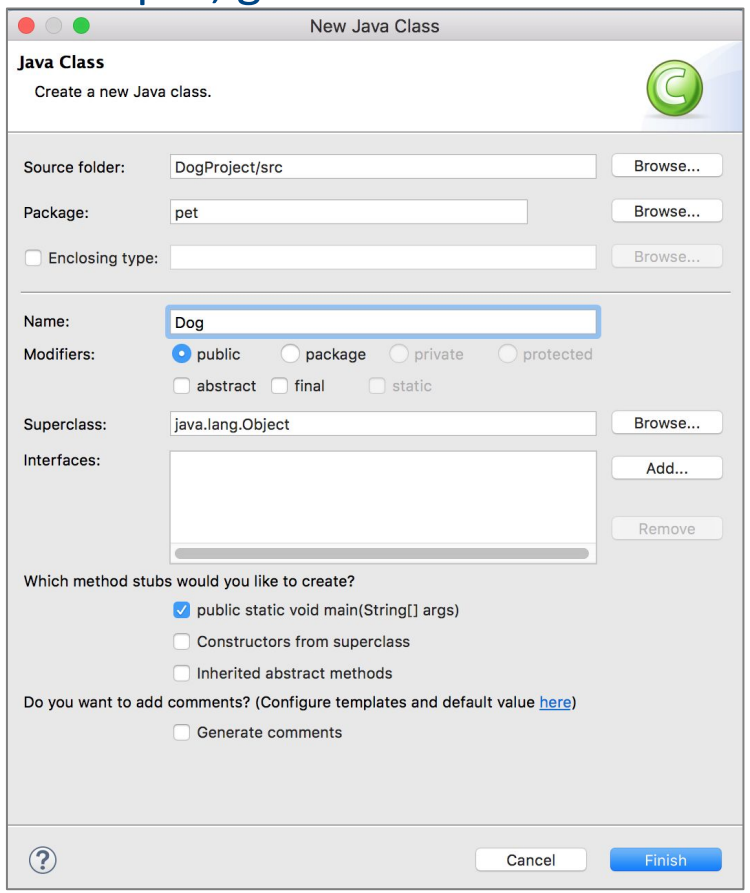

Make sure Create module-info.java file IS NOT checked

Use the default output folder

Click “Finish”

# Dog Class

- In Eclipse, go to “File” → “New” → “Class”




Provide a Package name
- Package names should not be capitalized

Provide a Class name
- Class names should be capitalized

Make sure public static void main(String[] args) IS checked

Make sure Inherited abstract methods IS NOT checked

Click “Finish”

# Dog Class

- The entry point of your Dog program is the *main* method

Dog.java

```
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 }
11
```

# Dog Class – Instance Variables

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

Dog.java

```
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

- 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
  - Again, the *Owner* data type doesn't exist yet.

```
41
42      //constructor
43
44      /**
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       */
51      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

- Create a new class Owner
  - This will represent a Dog owner

Dog.java | Owner.java

```
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 and other methods

```
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

- The *eat* method

```
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      }
```

# Dog Class - Methods

- The *haveBirthday* method

```
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      }
```

# Dog Class - Methods

- The *bark* method

```
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

- The *getDogInfo* method

```
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

```
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

```
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());
```

# Dog Class – *main* Method

```
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

```
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 Project

# Banking Project

- 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

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

Bank.java | BankAccount.java

```
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

Bank.java | BankAccount.java | Customer.java

```
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      }
```

# 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

```
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      }
```

# 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 account balance.
75       * @return rounded balance
76       */
77      public String getBalance() {
78          DecimalFormat df = new DecimalFormat("#.##");
79          df.setRoundingMode(RoundingMode.CEILING);
80
81          return df.format(this.balance);
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

```
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

```
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     }
```