1.1 Defining a class

Private fields

In addition to public member methods, a class definition has ***private fields***: variables that member methods can access but class users cannot. The private access modifier precedes each private field declaration.

PARTICIPATION ACTIVITY 1.1.1: Private fields.

Animation content:

```
There are two blue boxes containing code.

The first blue box contains the following code:
public class Restaurant {                        // Keeps a user's
review of a restaurant
   private String name;
   private int rating;

   public void setName(String restaurantName) {  // Sets the
restaurant's name
      ...
   }

   public void setRating(int userRating) {       // Sets the rating
(1-5, with 5 best)
      ...
   }

   public void print() {                         // Prints name and
rating on one line
      ...
   }
}
```

The following lines of code, from the first blue box, are highlighted with a white box:

```
   private String name;
   private int rating;
```

The second blue box contains the following code:

```
public class RestaurantFavorites {
   public static void main(String[] args) {
      Restaurant favLunchPlace = new Restaurant();

      favLunchPlace.rating = 5;
      ...
   }
}
```

The following portion of code, from the second blue box, is struck-through:

```
   favLunchPlace.rating
```


Animation captions:

1. A class definition has private fields for storing local data.
2. A class user cannot access a class' private fields; only the class' member methods can.

PARTICIPATION ACTIVITY 1.1.2: Private data members.

Consider the example above.

1) After creating a Restaurant object x, a class *user* can use a statement like x.name = "Sue's Diner".
   - True
   - False

2) After creating a Restaurant object x, a class user can use a statement like myString = x.name.
   - True
   - False

3) A class definition should provide comments along with each private field so that a class user knows how those fields are used.

- True
- False

Defining a class' public member methods

A programmer defining a class first names the class, *declares* private fields, and *defines* public member methods. A class' fields and methods are collectively called ***class members***.

The programmer *defines* the details of each member method, sometimes called the class' ***implementation***. A ***method definition*** provides an access modifier, return type, name, parameters, and the method's statements. A member method can access all private fields.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.1.1: A complete class definition, and use of that class.

Restaurant.java

```
public class Restaurant {                          // Info about a
restaurant
   private String name;
   private int rating;

   public void setName(String restaurantName) {    // Sets the
restaurant's name
      name = restaurantName;
   }

   public void setRating(int userRating) {         // Sets the rating
(1-5, with 5 best)
      rating = userRating;
   }

   public void print() {                           // Prints name and
rating on one line
      System.out.println(name + " -- " + rating);
   }
}
```


RestaurantFavorites.java

```
public class RestaurantFavorites {
   public static void main(String[] args) {
      Restaurant favLunchPlace = new Restaurant();
      Restaurant favDinnerPlace = new Restaurant();

      favLunchPlace.setName("Central Deli");
      favLunchPlace.setRating(4);

      favDinnerPlace.setName("Friends Cafe");
      favDinnerPlace.setRating(5);

      System.out.println("My favorite restaurants: ");
      favLunchPlace.print();
      favDinnerPlace.print();
   }
}
```

```
My favorite restaurants:
Central Deli -- 4
Friends Cafe -- 5
```


PARTICIPATION ACTIVITY 1.1.3: Class definition.

Consider the example above.

1) How is the name field declared?

- `private name;`
- `private String name;`
- `public String name;`

2) How does the print() member method's definition begin?

- `Restaurant.print();`
- `private void print()`
- `public void print()`

3) Which private fields of class Restaurant do the print() method's statements access?

- setName
- favLunchPlace and favDinnerPlace
- name and rating

Example: RunnerInfo class

The RunnerInfo class below maintains information about a person who runs, allowing a class user to set the time run and the distance run, and to get the runner's speed. The subsequent question set asks for the missing parts to be completed.

Figure 1.1.2: Simple class example: RunnerInfo.

RunnerInfo.java

```
class RunnerInfo {
   __(A)__ int timeRun;
   private double distRun;

   __(B)__ void setTime(int timeRunSecs) {      // Time run in seconds
      timeRun = timeRunSecs;                    // timeRun refers to the
field
   }

   public void setDist(double distRunMiles) {  // Distance run in miles
      distRun = distRunMiles;
   }

   public double getSpeedMph() {               // Speed in miles/hour
      return distRun / (timeRun / 3600.0);     // miles / (secs / (secs /
hour))
   }
}
```


RaceResults.java

```
public class RaceResults {
   public static void main(String[] args) {
      RunnerInfo runner1 = __(C)__;          // User-created object of
class type RunnerInfo
      RunnerInfo runner2 = new RunnerInfo(); // A second object

      runner1.setTime(360);
      runner1.setDist(1.2);
      runner2.setTime(200);
      runner2.setDist(0.5);

      System.out.println("Runner1's speed in MPH: " + runner1.__(D)__);
      System.out.println("Runner2's speed in MPH: " + __(E)__);
   }
}
```

```
Runner1's speed in MPH: 12
Runner2's speed in MPH: 9
```


PARTICIPATION ACTIVITY 1.1.4: Class example: RunnerInfo.

Complete the missing parts of the figure above.

1) (A)

Check Show answer

2) (B)

Check Show answer

3) (C)

Check Show answer

4) (D)

Check Show answer

5) (E)

Check Show answer

Exploring further:

- Classes from Oracle's Java tutorial.

CHALLENGE ACTIVITY 1.1.1: Classes.

Note: The below Java program consists of multiple source files. Click on the "CallPerson.java" or "Person.java" tabs to see the corresponding source file.

422352.2723990.qx3zqy7

Start

Type the program's output

CallPerson.java | Person.java

```
public class CallPerson {
   public static void main(String [] args) {
      String userName;
      Person person1 = new Person();

      userName = "Ron";

      person1.setFirstName(userName);
      System.out.print("He is " + person1.getFirstName());
   }
}
```

He is Ro

1 | 2 | 3

Check Next

CHALLENGE ACTIVITY 1.1.2: Basic class use.

Print person1's kids, call the incNumKids() method, and print again, outputting text as below. End each line with a newline.
Sample output for below program with input 3:

```
Kids: 3
New baby, kids now: 4
```

422352.2723990.qx3zqy7

```
1  // ===== Code from file PersonInfo.java =====
2  public class PersonInfo {
3     private int numKids;
4
5     public void setNumKids(int setPersonsKids) {
6        numKids = setPersonsKids;
7     }
8
9     public void incNumKids() {
10       numKids = numKids + 1;
11    }
12
13    public int getNumKids() {
14       return numKids;
15    }
```

Run

View your last submission

CHALLENGE ACTIVITY 1.1.3: Basic class definition.

Define the missing method. licenseNum is created as: (100000 * customID) + licenseYear, where customID is a method parameter. Sample output with inputs 2014 777:

```
Dog license: 77702014
```

422352.2723990.qx3zqy7

```
1  // ===== Code from file DogLicense.java =====
2  public class DogLicense {
3     private int licenseYear;
4     private int licenseNum;
5
6     public void setYear(int yearRegistered) {
7        licenseYear = yearRegistered;
8     }
9
10    // FIXME: Write createLicenseNum()
11
12    /* Your solution goes here  */
13
14    public int getLicenseNum() {
15       return licenseNum;
16    }
17 }
```

Run

View your last submission

1.2 Mutators, accessors, and private helpers

Mutators and accessors

A class' public methods are commonly classified as either mutators or accessors.

- A ***mutator*** method may modify ("mutate") a class' fields.
- An ***accessor*** method accesses fields but may not modify a class' fields.

Commonly, a field has two associated methods: a mutator for setting the value, and an accessor for getting the value, known as a ***setter*** and ***getter*** method, respectively, and typically with names starting with set or get. Other mutators and accessors may exist that aren't associated with just one field, such as the print() method below.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.2.1: Mutator and accessor methods.

Restaurant.java

```
public class Restaurant {
   private String name;
   private int rating;

   public void setName(String restaurantName) {  //
Mutator
      name = restaurantName;
   }

   public void setRating(int userRating) {       //
Mutator
      rating = userRating;
   }

   public String getName() {  // Accessor
      return name;
   }

   public int getRating() {  // Accessor
      return rating;
   }

   public void print() {      // Accessor
      System.out.println(name + " -- " + rating);
   }
}
```


MyRestaurant.java

```
public class MyRestaurant {
  public static void main(String[] args) {
     Restaurant myPlace = new Restaurant();
     myPlace.setName("Maria's Diner");
     myPlace.setRating(5);
     System.out.print(myPlace.getName() + " is rated
");
     System.out.println(myPlace.getRating());
  }
}
```

```
Maria's Diner is rated 5
```


PARTICIPATION ACTIVITY 1.2.1: Mutators and accessors.

1) A mutator should not change a class' private fields.

- True
- False

2) An accessor should not change a class' private fields.
   - True
   - False

3) A private field sometimes has a pair of associated set and get methods.
   - True
   - False

4) A getter method that changes a private field's value generates a compiler error.
   - True
   - False

Private helper methods

A programmer commonly creates private methods, known as ***private helper methods***, to help public methods carry out tasks.

PARTICIPATION ACTIVITY 1.2.2: Private helper member methods.

Animation content:

```
There are two blue boxes containing code.

The blue box on the left contains the following code:
public class MyClass {
   private int numA;

   private int methodX() {
      ...
   }
```

```
   public void method1() {
      numA = methodX();
      ...
   }
}
```


In the following line of code, from the blue box on the left, "methodX();" is highlighted with a white box with a blue outline.

```
      numA = methodX();
```

Next to the code "methodX();" in the blue box on the left is the text "OK" in the color blue.

The blue box on the right contains the following code:

```
public class TestClass {
   public static void main(String[] args) {

      MyClass someObj = new MyClass();

      someObj.method1();

      ...

      someObj.methodX();
   }
}
```

The following line of code from the blue box on the right is highlighted with a white box with a blue outline:

```
      someObj.method1();
```

Next to the code "someObj.method1();" in the blue box on the right is the text "OK" in the color blue.

The following line of code from the blue box on the right is highlighted with a white box with a red outline:

```
      someObj.methodX();
```

Next to the code "someObj.methodX();" in the blue box on the right is

```
the text "Error" in the color red.
```

Animation captions:

1. In addition to public member methods, a class may define private member methods.
2. Any member method (public or private) may call a private member method.
3. A user of the class can call public member methods, but a user can not call private member methods (which would yield a compiler error).

PARTICIPATION ACTIVITY 1.2.3: Private helper methods.

1) A class' private helper method can be called from main(), which is defined in another class.
   - ○ True
   - ○ False

2) A private helper method typically helps public methods carry out their tasks.
   - ○ True
   - ○ False

3) A private helper method cannot call another private helper method.
   - ○ True
   - ○ False

4) A public member method may not call another public member method.
   - ○ True
   - ○ False

CHALLENGE ACTIVITY 1.2.1: Mutators, accessors, and private helpers.

422352.2723990.qx3zqy7

Start

Type the program's output

CallDog.java | Dog.java

```
public class CallDog {
   public static void main(String [] args) {
      Dog buddy = new Dog();

      buddy.setAge(54);
      System.out.print(buddy.getStage());

   }
}
```

Adulthood

1 | 2 | 3

Check Next

1.3 Constructor overloading

Basics

Programmers often want to provide different initialization values when creating a new object. A class creator can ***overload*** a constructor by defining multiple constructors differing in parameter types. When an object is created with the new operator, arguments can be passed to the constructor. The constructor with matching parameters will be called.

PARTICIPATION ACTIVITY 1.3.1: Overloaded constructors.

Animation content:

Shows code having two constructors, one with no parameters, and one with two parameters. Then in main(), one object creation has no arguments, and another has two arguments.

Animation captions:

1. Creating a new object with no constructor arguments calls the default constructor. In this case, the object gets initialized with NoName and -1.
2. Passing a String and int argument to the constructor causes the constructor with matching

parameters to be called instead. The object gets initialized with those argument values.

zyDE 1.3.1: Overloading a constructor.

Current file: Restaurant.java ▾ Load default ter


```
1  public class Restaurant {
2     private String name;
3     private int rating;
4
5     // Default constructor
6     public Restaurant() {
7        name = "NoName";
8        rating = -1;
9     }
10
11    // Another constructor
12    public Restaurant(String initName, int initRating) {
13       name = initName;
14       rating = initRating;
15    }
16
17    // Prints name and rating on one line
```

Run

PARTICIPATION ACTIVITY 1.3.2: Overloaded constructors.

Given the three constructors below, indicate which will be called for each object creation.

```
public class SomeClass {
   public SomeClass() { ... }                      // A
   public SomeClass(String name) { ... }           // B
   public SomeClass(String name, int num) { ... } // C
}
```


1) `SomeClass myObj = new SomeClass("Lee");`

- ○ A
- ○ B
- ○ C

- Error

2) `SomeClass myObj = new SomeClass();`

- A
- B
- Error

3) `SomeClass myObj = new SomeClass("Lee", 5, 0);`

- C
- Error

If any constructor defined, should define default

If a programmer defines any constructor, the compiler does not implicitly define a default constructor, so good practice is for the programmer to also explicitly define a default constructor so that an object creation like `new MyClass()` remains supported.

Figure 1.3.1: Error - The programmer defined a constructor, so the compiler does not automatically define a default constructor.

```
public class Restaurant {
   public Restaurant(String initName, int initRating) {
      ...
   }

   // No other constructors
}

public class RestaurantFavorites {
   public static void main(String[] args) {
      Restaurant foodPlace = new Restaurant();
      ...
   }
}
```



```
RestaurantFavorites.java:3: cannot find symbol
symbol  : constructor Restaurant()
location: class Restaurant
      Restaurant foodPlace = new Restaurant();
                             ^
```

PARTICIPATION ACTIVITY 1.3.3: Constructor definitions.

Which of the following is OK as the entire set of constructors for public class MyClass? Assume object creation like `MyClass x = new MyClass();` should be supported.

1) `public MyClass() { ... }`

- OK
- Error

2) `// None`

- OK
- Error

3)
```
public MyClass() { ... }
public MyClass(String name) {
... }
```

- OK
- Error

4)
```
public MyClass(String name) {
... }
```

- OK
- Error

CHALLENGE ACTIVITY 1.3.1: Enter the output of the constructor overloading.

422352.2723990.qx3zqy7

Start

Type the program's output

CallPet.java | Pet.java

```
Unnamed, -1
Cleo, 5
```

```
public class CallPet {
   public static void main(String[] args) {
      Pet dog = new Pet();
      Pet cat = new Pet("Cleo", 5);

      dog.print();
      cat.print();
   }
}
```


CHALLENGE ACTIVITY 1.3.2: Constructor overloading.

Write a second constructor as indicated. Sample output:

```
User1: Minutes: 0, Messages: 0
User2: Minutes: 1000, Messages: 5000
```

422352.2723990.qx3zqy7

```
1  // ===== Code from file PhonePlan.java =====
2  public class PhonePlan {
3     private int freeMinutes;
4     private int freeMessages;
5
6     public PhonePlan() {
7         freeMinutes  = 0;
8         freeMessages = 0;
9      }
10
11     // FIXME: Create a second constructor with numMinutes and numMessages parameters.
12
13    /* Your solution goes here  */
14
15    public void print() {
16       System.out.println("Minutes: " + freeMinutes + ", Messages: " + freeMessages);
17    }
```

Run

View your last submission

1.4 Objects and references

References

A ***reference*** is a variable type that refers to an object. A reference may be thought of as storing the memory address of an object. Variables of a class data type (and array types, discussed elsewhere) are reference variables.

PARTICIPATION ACTIVITY 1.4.1: A simple illustration of references and objects.

Animation content:

A code snippet is given to the left:

```
// Reference variable does not refer
// to any object upon definition
TimeHrMin travelTime;

// New allocates memory for object, returns
// reference to object
travelTime = new TimeHrMin();

travelTime.setHour(2);
travelTime.setMinute(40);

travelTime.printTime();
```

To the right is a color key for memory. A blue location in the stack means "memory for TimeHrMin object instance." A yellow location in the stack means "memory for reference variable."

Below is the color coded stack. There are four locations, labeled 94 to 97. Location 94 is yellow and contains the value 96, which is the travelTime variable. Location 95 is empty and is a neutral color. Location 96 is blue and contains the value 2, which is the hrVal variable. Location 97 is also blue and contains the value 40, which is the minVal variable.

Underneath is a computer screen with the output of the program:
2 hour(s) and 40 minute(s)

Animation captions:

1. TimeHrMin travelTime; declares a reference variable of type TimeHrMin. No TimeHrMin

object is yet created.
2. travelTime = new TimeHrMin(); creates a new TimeHrMin object at memory address 96. travelTime refers to that object.
3. The setHour() method assigns 2 to the hrVal field of the TimeHrMin object referred to by travelTime. The setMinute() method assigns 40 to the minVal field of the TimeHrMin object referred to by travelTime.
4. travelTime.printTime(); calls the printTime() method on the TimeHrMin object referred to by travelTime, which prints 2 hour(s) and 40 minute(s)

A statement like `TimeHrMin travelTime;` declares a reference to an object of type TimeHrMin, while `String firstName;` declares a reference to an object of type String. The reference variables do not store data for those class types. Instead, the programmer must assign each reference to an object, which can be created using the `new` operator.

The statement `TimeHrMin travelTime;` declares a reference variable with an unknown value. A common error is to attempt to use a reference variable that does not yet refer to a valid object.

The ***new*** operator allocates memory for an object, then returns a reference to the object's location in memory. Thus, `travelTime = new TimeHrMin();` sets travelTime to refer to a new TimeHrMin object in memory. travelTime now refers to a valid object and the programmer may use travelTime to access the object's methods. The reference variable declaration and object creation may be combined into a single statement: `TimeHrMin travelTime = new TimeHrMin();`

Java does not provide a direct way to determine the memory location of an object, or to determine the exact address to which a reference variable refers. The "value" of a reference variable is unknown to the programmer. This material's animations show the memory address of an object as the value for a reference variable for illustrative purposes, to illustrate that a reference variable and its object are separate entities in memory.

PARTICIPATION ACTIVITY 1.4.2: Referring to objects.

1) Declare a reference variable named flightPlan that can refer to an object of type FlightInfo. Do not create a new object.

Check Show answer

2) Write a statement that creates an object of FlightInfo and assigns the new object to the

reference variable flightPlan.

Check Show answer

Multiple object references

Two or more reference variables may refer to the same object, as illustrated below.

PARTICIPATION ACTIVITY 1.4.3: Multiple reference variables may refer to the same object.

Animation content:

A code snippet is given to the left:

```
RunnerInfo lastRun;
RunnerInfo currRun = new RunnerInfo();

currRun.setTime(300);
currRun.setDist(1.5);

System.out.print("Run speed: ");
System.out.println(currRun.getSpeed());

// Assign reference to lastRun
lastRun = currRun;

lastRun.setTime(250);
lastRun.setDist(1.2);
```

To the right is a memory stack. There are five locations, labeled 96 to 100. Location 96 contains the value 99, which is the lastRun variable. Location 97 contains the value 99, which is the currRun variable. Location 98 is empty. Location 99 contains the value 250, which is the timeRun variable. Location 100 contains the value 1.2, which is the distRun variable.

Underneath is a computer screen with the output of the program:

```
Run speed: 18
```

Animation captions:

1. A reference variable lastRun of type RunnerInfo is declared but not initialized. A reference variable currRun of type RunnerInfo is declared and initialized to a new RunnerInfo object.
2. currRun's time is set to 300 and distance is set to 1.5. The getSpeed() method on the RunnerInfo object referred to by currRun returns the run speed.
3. Assigning lastRun with currRun causes lastRun and currRun to refer to the same object.
4. Calling the setTime() and setDist() methods on lastRun updates the objects referred to by both lastRun and currRun.

PARTICIPATION ACTIVITY 1.4.4: Multiple object references.

Refer to the following code:

```
DriveTime timeRoute1 = new DriveTime();
DriveTime timeRoute2;
DriveTime bestRoute;

timeRoute2 = new DriveTime();
bestRoute = timeRoute1;
```

1) Variables timeRoute1 and timeRoute2 both refer to valid objects.
   - True
   - False

2) Variables timeRoute1 and bestRoute refer to the same object.
   - True
   - False

1.5 The 'this' implicit parameter

Implicit parameter

An object's member method is called using the syntax `objectReference.method()`. The

object reference before the method name is known as an ***implicit parameter*** of the member method because the compiler converts the call syntax `objectReference.method(...)` into a method call with the object reference implicitly passed as a parameter. Ex: `method(objectReference, ...)`.

Within a member method, the implicitly-passed object reference is accessible via the keyword ***this***. In particular, a class member can be accessed as `this.classMember`. The "." is the member access operator.

Using `this` makes clear that a class member is being accessed and is essential if a field member and parameter have the same identifier. In the example below, `this` is necessary to differentiate between the field member sideLength and the parameter sideLength.

Figure 1.5.1: Using 'this' to refer to an object's members.

ShapeSquare.java:

```
public class ShapeSquare {
   // Private fields
   private double sideLength;

   // Public methods
   public void setSideLength(double sideLength) {
      this.sideLength = sideLength;
      // Field member    Parameter
   }

   public double getArea() {
      return sideLength * sideLength; // Both refer to
field
   }
}
```

ShapeTest.java:

```
public class ShapeTest {
   public static void main(String[] args) {
      ShapeSquare square1 = new ShapeSquare();

      square1.setSideLength(1.2);
      System.out.println("Square's area: " +
square1.getArea());
   }
}
```

```
Square's area:
1.44
```


PARTICIPATION ACTIVITY 1.5.1: The 'this' implicit parameter.

Given a class Spaceship with private field numYears and method:
`public void addNumYears(int numYears)`

1) In addNumYears(), which line assigns the field numYears with 0?

- `numYears = 0;`
- `this(numYears) = 0;`
- `this.numYears = 0;`

2) In addNumYears(), which line assigns the field numYears with the parameter numYears?

- `numYears = this.numYears;`
- `this.numYears = numYears;`

3) In addNumYears(), which line adds the parameter numYears to the existing value of field numYears?

- `this.numYears = this.numYears + numYears;`
- `this.numYears = numYears + numYears;`
- `numYears = this.numYears + numYears;`

4) Given variable `Spaceship ss1 = new Spaceship()` is declared in main(), which line assigns ss1's numYears with 5?

- `ss1.numYears = 5;`
- `ss1(numYears) = 5;`
- `this.numYears = 5;`
- None of the above.

Using 'this' in class member methods and constructors

The animation below illustrates how member methods work. When an object's member method is called, the object's reference, which can be thought of as the object's memory address, is passed to the method via the implicit "this" parameter. An access in setTime() to `this.hours` first goes to the object's address, then to the hours field.

PARTICIPATION ACTIVITY 1.5.2: How class member methods work.

Animation content:

There are two blue boxes containing code. To the right the memory is displayed graphically.

The top blue box contains the following code:

```
public class ElapsedTime {
   private int hours;
   private int minutes;

   public void setTime(int timeHr, int timeMin) {
      this.hours = timeHr;
      this.minutes = timeMin;
   }
}
```

The bottom blue box contains the following code:

```
public class SimpleThisKeywordEx {
   public static void main(String [] args) {
      ElapsedTime travTime = new ElapsedTime();
      int usrHrs;
      int usrMins;

      usrHrs = 5;
      usrMins = 34;

      travTime.setTime(usrHrs, usrMins);
   }
}
```

```
The memory contains the following:
Addresses 96-99 store memory for the main() method.
Address 96 contains the value of the instance variable hours, which is
part of the object travTime. Address 96 contains 5.
Address 97 contains the value of the instance variable minutes, which
is part of the object travTime. Address 97 contains 34.
Address 98 contains the value of the variable userHrs. Address 98
contains 5.
Address 99 contains the value of the variable userMins. Address 99
contains 34.

Addresses 101-103 store memory for travTime's SetTime() member method.
Address 101 is the value contained by the "this" keyword, which is
travTime's memory address. Address 101 contains the value 96.
Address 102 contains the value of the variable timeHr. Address 102
contains 5.
Address 103 contains the value of the variable timeMin. Address 103
contains 34.
```


Animation captions:

1. travTime is an object of class type ElapsedTime.
2. When travTime's SetTime() member method is called, travTime's object reference, which can be thought of as the object's memory address, is passed to the method via the implicit 'this' parameter.
3. The implicitly-passed object reference is accessible within the member method via the keyword 'this'. Ex: this.hours first goes to travTime's memory address, then to the hours data member.

The "this" keyword can also be used in a constructor to invoke a different (overloaded) constructor. In the default constructor below, `this(0, 0);` invokes the other constructor to initialize both fields to zero. For this example, a programmer could have just set both fields to zero within the default constructor. However, invoking other constructors is useful when a class' initialization routine is lengthy and avoids rewriting the same code.

Figure 1.5.2: Calling overloaded constructor using this keyword.

```
public class ElapsedTime {
   private int hours;
   private int minutes;

   // Overloaded constructor definition
   public ElapsedTime(int timeHours, int
timeMins) {
      hours   = timeHours;
      minutes = timeMins;
   }

   // Default constructor definition
   public ElapsedTime() {
      this(0, 0);
   }

   // Other methods ...
}
```


PARTICIPATION ACTIVITY 1.5.3: Using the 'this' keyword in member methods and constructors.

1) Write a statement that uses the "this" keyword to initialize the field minutes to 0.

Check Show answer

2) Using the ElapsedTime class declared above, complete the default constructor so that the hours and minutes fields are both initialized to -1 by making a call to the overloaded constructor.

```
public ElapsedTime() {
    ;
}
```

Check Show answer

Exploring further:

- Using the this keyword from Oracle's Java tutorial.

CHALLENGE ACTIVITY 1.5.1: Enter the output for the this operator.

422352.2723990.qx3zqy7

Start

Type the program's output

CallAirplane.java | Airplane.java

```
public class CallAirplane {
   public static void main(String[] args) {
      Airplane boeing747 = new Airplane();

      boeing747.setSpeed(800);
      boeing747.print();
   }
}
```

```
800 MPH
```

1 | 2

Check Next

CHALLENGE ACTIVITY 1.5.2: The this implicit parameter.

Define the missing method. Use "this" to distinguish the local member from the parameter name.

422352.2723990.qx3zqy7

```
1 // ===== Code from file CablePlan.java =====
2 public class CablePlan {
3    private int numDays;
4
5    // FIXME: Define setNumDays() method, using "this" implicit parameter.
```

Run

View your last submission

1.6 Primitive and reference types

Wrapper classes

Java variables are one of two types.

- A ***primitive type*** variable directly stores the data for that variable type, such as int, double, or char. Ex: `int numStudents = 20;` declares an int that directly stores the data 20.
- A ***reference type*** variable can refer to an instance of a class, also known as an object.

Java provides several ***wrapper classes*** that are built-in reference types that augment the primitive types. The ***Integer*** data type is a built-in class in Java that augments the int primitive type. Ex: `Integer maxPlayers = 10;` declares an Integer reference variable named maxPlayers that refers to an instance of the Integer class, also known as an Integer object. That Integer object stores the integer value 10.

Many of Java's built-in classes, such as Java's Collection library, only work with objects. For example, a programmer can create an ArrayList containing Integer elements, e.g., `ArrayList<Integer> frameScores;` but not an ArrayList of int elements. Wrapper classes allow the program to create objects that store a single primitive type value, such as an integer or floating-point value. The wrapper classes also provide methods for converting between primitive types (e.g., int to double), between number systems (e.g., decimal to binary), and between a primitive type and a String representation.

Table 1.6.1: Commonly used wrapper classes.

Reference type	Associated primitive type
Character	char
Integer	int
Double	double
Boolean	boolean
Long	long

PARTICIPATION ACTIVITY 1.6.1: Identifying usage of wrapper classes.

Determine whether each statement uses a primitive type or a wrapper class.

1) `Integer playerCount = 10;`
   - ○ Wrapper class
   - ○ Primitive type

2) `long uniqueID = 1000L;`
   - ○ Wrapper class
   - ○ Primitive type

3) `Character parseLetter = 'A';`
   - ○ Wrapper class
   - ○ Primitive type

4) `ArrayList<Double> times = new ArrayList<Double>();`
   - ○ Wrapper class
   - ○ Primitive type

Memory allocation for wrapper class objects

A programmer may use a wrapper class variable in expressions in the same manner as the primitive type int. An expression may even combine Integers, ints, and integer literals.

A wrapper class object (as well as a String object) is ***immutable***, meaning a programmer cannot change the object via methods or variable assignments after object creation. When the result of an expression is assigned to an Integer reference variable, memory for a new Integer object with the computed value is allocated, and the reference (or address) of this new object is assigned to the reference variable. A new memory allocation occurs every time a new value is assigned to an Integer variable, and the previous memory location to which the variable referred, remains unmodified.

PARTICIPATION ACTIVITY 1.6.2: Time calculation using Integer class.

Animation content:

```
Code is written in a blue box on the left. To the right the memory is
displayed graphically.

The code in the blue box on the left is as follows:
int timeMins = 0;
Integer timeHrs = 0;

timeMins = 400;
timeHrs = timeMins / 60;

The memory is as follows:
Address 94 contains the value 400. Address 94 holds the value for the
variable timeMins.
Address 95 contains the value 101. Address 95 holds the value for the
variable timeHrs.
Address 98 contains the value 0. Address 98 holds the value for an
Integer object.
Address 101 contains teh value 6. Address 101 holds the value for an
Integer object.
```


Animation captions:

1. timeMins is a primitive type variable. A primitive type variable directly stores the data for that variable type.
2. Assigning the Integer variable timeHrs with the literal 0 assigns timeHrs with a reference to

an Integer object with value 0. Note that Java maintains a cache of Integer objects for literal values -128 to 127 (inclusive).

3. timeMins is a primitive type and directly stores the value 400.
4. A new Integer object is created and assigned with 400 / 60, or 6, and timeHrs is updated to refer to that new Integer object.

PARTICIPATION ACTIVITY 1.6.3: Memory contents of primitive and wrapper class variables.

Indicate the type of value that each variable in the following program is assigned.

```
Double timeRemaining = 1.3;
char quitInput = 'q';
Integer loopIndex = 0;
Boolean shouldExit = false;
boolean isFound = true;
```

If unable to drag and drop, refresh the page.

Primitive boolean value **Reference to Integer object** **Reference to Boolean object**

Reference to Double object **Primitive char value**

quitInput

shouldExit

isFound

timeRemaining

loopIndex

Reset

Example: Fly Drive

The following example uses the ***Double*** class to calculate the amount of time necessary to drive or fly a certain distance.

Figure 1.6.1: Program using the Double class to calculate flight and driving times.

```
import java.util.Scanner;

public class FlyDrive {
   public static void main(String [] args) {
      Scanner scnr = new Scanner(System.in);
      Double distMiles;
      Double hoursFly;
      Double hoursDrive;

      System.out.print("Enter a distance in miles: ");
      distMiles = scnr.nextDouble();

      hoursFly = distMiles / 500.0;
      hoursDrive = distMiles / 60.0;

      System.out.println(distMiles + " miles would take:");
      System.out.println(hoursFly + " hours to fly");
      System.out.println(hoursDrive + " hours to drive");
   }
}
```

```
Enter a distance in miles: 450
450.0 miles would take:
0.9 hours to fly
7.5 hours to drive
...
Enter a distance in miles: 20.5
20.5 miles would take:
0.041 hours to fly
0.3416666666666667 hours to drive
```


Limits on initialization values

When using a literal for initialization, the programmer must ensure that the literal's value falls within the appropriate numeric range, e.g., -2,147,483,648 to 2,147,483,647 for an integer. The wrapper classes (except for Character and Boolean) declare the MAX_VALUE and MIN_VALUE fields, which are static fields initialized with the maximum and minimum values a type may represent, respectively. A programmer may access these fields to check the supported numeric range by typing the wrapper class' name followed by a dot and the field name, as in ***`Integer.MIN_VALUE`****, which returns -2,147,483,648.*

PARTICIPATION ACTIVITY 1.6.4: Declaring wrapper class objects.

1) Declare a variable called gameScore that refers to a wrapper class object with an int value of 81.

Check **Show answer**

2) Declare a variable called trackCircumference that refers to a wrapper class object with a double value of 29.5.

Check **Show answer**

3) Declare a variable called logoChar that refers to a wrapper class object with a char value of 'U'.

Check **Show answer**

Comparing wrapper class objects

For reference variables of wrapper classes (e.g., Integer, Double, Boolean), a common error is to use the equality operators == and != when comparing values, which does not work as expected. Using the equality operators on any two reference variables evaluates to either true or false depending on each operand's referenced object. For example, given two Integers num1 and num2, the expression `num1 == num2` compares if both num1 and num2 reference the same Integer object, but does not compare the Integers' contents. Because those references will (usually) be different, `num1 == num2` will evaluate to false. This is not a syntax error, but clearly a logic error.

Although a programmer should never compare two reference variables of wrapper classes using the equality operators, a programmer may use the equality operators when comparing a wrapper class object with a primitive variable or a literal constant. The relational operators <, <=, >, and >= may be used to compare wrapper class objects. However, note that relational operators are not typically valid for other reference types. The following table summarizes allowable comparisons.

Table 1.6.2: Comparing wrapper class objects using relational operators.

`objectVar == objectVar` (also applies to !=)	DO NOT USE. Compares references to objects, not the value of the objects.
`objectVar == primitiveVar` (also applies to !=)	OK. Compares value of object to value of primitive variable.
`objectVar == 100` (also applies to !=)	OK. Compares value of object to literal constant.
`objectVar < objectVar` (also applies to <=, >, and >=)	OK. Compares values of objects.
`objectVar < primitiveVar` (also applies to <=, >, and >=)	OK. Compares values of object to value of primitive.
`objectVar < 100` (also applies to <=, >, and >=)	OK. Compares values of object to literal constant.

Reference variables of wrapper classes can also be compared using the ***equals()*** and ***compareTo()*** methods. These method descriptions are presented for the Integer class, but apply equally well to the other wrapper classes. Although the use of comparison methods is slightly cumbersome in comparison to relational operators, these comparison methods may be preferred by programmers who do not wish to memorize exactly which comparison operators work as expected.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Table 1.6.3: equals() and compareTo() methods for wrapper class types.

Given:
`Integer num1 = 10;`
`Integer num2 = 8;`
`Integer num3 = 10;`
`int regularInt = 20;`	

equals(otherInt)	true if both Integers contain the same value. otherInt may be an Integer object, int variable, or integer literal.
	
	`num1.equals(num2) // Evaluates to false`
	`num1.equals(10)  // Evaluates to true`
	`!(num2.equals(regularInt)) // Evaluates to true because 8 != 20`

compareTo(otherInt)	return 0 if the two Integer values are equal, returns a negative number if the Integer value is less than otherInt's value, and returns a positive number if the Integer value is greater than otherInt's value. otherInt may be an Integer object, int variable, or integer literal.
	
	`num1.compareTo(num2) // Returns value greater than 0, because 10 > 8`
	`num2.compareTo(8)  // Returns 0 because 8 == 8`
	`num1.compareTo(regularInt)  // Returns value less than 0, because 10 < 20`

PARTICIPATION ACTIVITY 1.6.5: Comparing wrapper class objects.

Given the following wrapper class objects, determine whether the provided comparisons evaluate to true or false:

```
Integer score1 = 195;
Integer score2 = 191;
Integer score3 = 195;
int maxScore = score1;
```


1) score1 < score2

- True
- False

2) score1 <= score3

- True
- False

3) score2 < maxScore

- True
- False

4) score1 == score3

- True
- False

5) 198 < score3

- True
- False

6) score1.equals(score3)

- True
- False

7) score2.compareTo(score1) > 0

- True
- False

Exploring further: Links to Oracle's Java specification for wrapper classes:

- Number
- Character
- Boolean

1.7 Static fields and methods

Static fields

The keyword ***static*** indicates a variable is allocated in memory only once during a program's execution. Static variables reside in the program's static memory region and have a global scope. Thus, static variables can be accessed from anywhere in a program.

In a class, a ***static field*** is a field of the class instead of a field of each class object. Thus, static fields are independent of any class object, and can be accessed without creating a class object. Static fields are declared and initialized in the class definition. Within a class method, a static field is accessed using the field name. A public static field can be accessed outside the class using dot notation: ClassName.fieldName.

Instance and class variable

*Static fields are also called **class variables**, and non-static fields are also called **instance variables**.*

PARTICIPATION ACTIVITY 1.7.1: Static field used to create object ID numbers.

Animation content:

undefined

Animation captions:

1. The Store class' static field nextId is declared and initialized in the Store class definition. Only one instance of the field nextId will exist in memory.
2. When a Store object is created, memory is allocated for the object's name, type, and id fields, but not the static field nextId.
3. The constructor assigns an object's id with nextId, and then increments nextId. Each time an object is created, nextId is incremented. Thus, each object will have a unique id.
4. Any class method can access or mutate a static field. Because nextId is public, nextId can also be accessed outside the class using the member access operator (.).

PARTICIPATION ACTIVITY 1.7.2: Static field.

1) Each constructed class object creates a new instance of a static field.

- True
- False

2) All static fields can be accessed anywhere in a program.

- True
- False

3) A public static field can be accessed outside of the class using dot notation.

- True
- False

Static member methods

A ***static member method*** is a class method that is independent of class objects. Static member methods are typically used to access and mutate private static fields from outside the class. Since static methods are independent of class objects, the `this` parameter is not passed to a static member method. So, a static member method can only access a class' static fields.

Figure 1.7.1: Static member method used to access a private static field.

Store.java

```
public class Store {
   // Declare and initialize private static field
   private static int nextId = 101;

   // Define private fields
   private String name;
   private String type;
   private int id;

   public Store(String storeName, String storeType) {
      name = storeName;
      type = storeType;
      id = nextId;

      ++nextId;   // Increment each time a Store object is created
   }

   public int getId() {
      return id;
   }

   public static int getNextId() {
      return nextId;
   }
}
```


NewStores.java

```
public class NewStores {
   public static void main(String[] args) {
      Store store1 = new Store("Macy's", "Department");
      Store store2 = new Store("Albertsons", "Grocery");
      Store store3 = new Store("Ace", "Hardware");

      System.out.println("Store 1's ID: " + store1.getId());
      System.out.println("Store 2's ID: " + store2.getId());
      System.out.println("Store 3's ID: " + store3.getId());
      System.out.println("Next ID: " + Store.getNextId());
   }
}
```



```
Store 1's ID: 101
Store 2's ID: 102
Store 3's ID: 103
Next ID: 104
```

PARTICIPATION ACTIVITY 1.7.3: Static methods.

1) A static method is needed to access or mutate a _____ static field from outside of the class.

- public
- private

2) The `this` parameter can be used in a static method to access an object's non-static fields.

- True
- False

CHALLENGE ACTIVITY 1.7.1: Enter the output with static members.

422352.2723990.qx3zqy7

Start

Type the program's output

CallFood.java | FoodType.java

```
public class CallFood {
   public static void main(String[] args) {
      FoodType order1 = new FoodType("Soup");
      FoodType order2 = new FoodType("Cake");
      FoodType order3 = new FoodType("Muffins");

      order1.print();
      order2.print();
      order3.print();
   }
}
```

```
0: Soup
2: Cake
4: Muffins
```

1

Check Next

1.8 Strings

Strings and string literals

A ***string*** is a sequence of characters. A ***string literal*** surrounds a character sequence with double quotes, as in "Hello", "52 Main St.", or "42", vs. an integer literal like 42 or character literal like 'a'. Various characters may be in a string, such as letters, numbers, spaces, or symbols like $ or %, as in "$100 for Julia!!". Earlier sections showed string literals being output, as in: `System.out.print("Hello");`.

PARTICIPATION ACTIVITY 1.8.1: A string is stored as a sequence of characters in memory.

Type a string to see how a string is stored as a sequence of characters in memory. In this case, the string happens to be in memory locations 501 to 506. Try typing Hello! or 627.

Type a string (up to 6 characters): `Julia`

	Memory
501	**J**
502	**u**
503	**l**
504	**i**
505	**a**
506	

PARTICIPATION ACTIVITY 1.8.2: String literals.

Indicate which items are string literals.

1) "Hey"

- Yes
- No

2) "Hey there."

- Yes
- No

3) 674

- Yes
- No

4) "674"

- Yes
- No

5) 'ok'

- Yes
- No

6) "a"

- Yes
- No

String variables and assignments

Some variables should hold a string. A programmer declares a string variable similarly to declaring char, int, or double variables, by using the String data type. Note the capital S.

A String variable is a reference type (discussed in depth elsewhere) variable that refers to a String object. An ***object*** consists of some internal data items plus operations that can be performed on that data. Ex: `String movieTitle = "Frozen";` declares a String reference variable named movieTitle that refers to a String object. That String object stores the string "Frozen".

A programmer can assign a String variable with a string literal, which creates a new String object. Ex: `movieTitle = "The Martian";` creates a new String object with the string "The Martian", and assigns the String object's reference to the variable movieTitle.

Assigning one String variable to another String variable causes both variables to refer to the same String, and does not create a new String. Ex: `movieTitle = favoriteMovie;` assigns favoriteMovie's reference to movieTitle. Both movieTitle and favoriteMovie refer to the same String object.

A programmer can initialize a string variable during declaration, as in `String firstMonth = "January";`.

Figure 1.8.1: Declaring and assigning a string.

```
public class SentenceFromStrings {
   public static void main(String[] args) {
      String sentenceSubject;
      String sentenceVerb;
      String sentenceObject = "an apple";

      sentenceSubject = "boy";
      sentenceVerb = "ate";

      System.out.print("A ");
      System.out.print(sentenceSubject + "
");
      System.out.print(sentenceVerb + " ");
      System.out.println(sentenceObject +
".");
   }
}
```



```
A boy ate an
apple.
```

PARTICIPATION ACTIVITY 1.8.3: Declaring and assigning a String variable.

1) Declare a string variable userName.

Check Show answer

2) Write a statement that assigns userName with "Sarah".

Check Show answer

3) Suppose string str1 is initially "Hello" and str2 is "Hi". After `str1 = str2;`, what is str1?
Omit the quotes.

Check Show answer

4) Suppose string str1 is initially "Hello" and str2 is "Hi". After `str1 = str2;` and then `str2 = "Bye";`, what is str1? Omit the quotes.

Check Show answer

5) Type one statement that declares a string named smallestPlanet, initialized with "Mercury".

Check Show answer

Getting a string without whitespaces from input

A ***whitespace character*** is a character used to represent horizontal and vertical spaces in text, and includes spaces, tabs, and newline characters. Ex: "Oh my goodness!" has two whitespace characters, one between h and m, the other between y and g.

Below shows the basic approach to get a string from input into variable userString. The approach automatically skips initial whitespace, then gets characters until the next whitespace is seen.

```
userString = scnr.next();
```

PARTICIPATION ACTIVITY 1.8.4: Getting a string without whitespace from input.

For the given input, indicate what string will be put into userString by:


```
userString = scnr.next();
```

1) abc

Check Show answer

2) Hi there.

Check Show answer

3) Hello! I'm tired.

Check Show answer

4) Very fun.

Check Show answer

PARTICIPATION ACTIVITY 1.8.5: Getting a string without whitespace from input (continued).

For the given input, indicate what string will be put into secondString by:

```
firstString = scnr.next();
secondString = scnr.next();
```

1) Oh my!

Check Show answer

2) Frank
Sinatra

Check Show answer

3) Oh...

...no!

Check Show answer

4) We all know

Check Show answer

Example: Word game

The following example illustrates getting strings from input and putting strings to output.

Figure 1.8.2: Strings example: Word game.

```
import java.util.Scanner;

/* A game inspired by "Mad Libs" where user enters
nouns,
   verbs, etc., and then a story using those words is
output.
*/

public class StoryGame {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String wordRelative;
      String wordFood;
      String wordAdjective;
      String wordTimePeriod;

      // Get user's words
      System.out.println("Provide input without
spaces.");

      System.out.println("Enter a kind of relative:
");
      wordRelative = scnr.next();

      System.out.println("Enter a kind of food: ");
      wordFood = scnr.next();

      System.out.println("Enter an adjective: ");
      wordAdjective = scnr.next();

      System.out.println("Enter a time period: ");
      wordTimePeriod = scnr.next();

      // Tell the story
      System.out.println();
      System.out.print  ("My " + wordRelative);
      System.out.println(" says eating " + wordFood);
      System.out.println("will make me more " +
wordAdjective + ",");
      System.out.println("so now I eat it every " +
wordTimePeriod + ".");
   }
}
```

```
Provide input
without spaces.
Enter a kind of
relative:
mother
Enter a kind of
food:
apples
Enter an adjective:
loud
Enter a time period:
week

My mother says
eating apples
will make me more
loud,
so now I eat it
every week.
```


Getting a string with whitespace from input

Sometimes a programmer wishes to get whitespace characters into a string, such as getting a user's input of the name "Franklin D. Roosevelt" into a string variable presidentName.

For such cases, the language supports getting an entire line into a string. The method ***scnr.nextLine()*** gets all remaining text on the current input line, up to the next newline character (which is removed from input but not put in stringVar).

PARTICIPATION ACTIVITY 1.8.6: Getting a string with whitespace from input.

What does the following statement store into the indicated variable, for the given input?

```
firstString = scnr.nextLine();
secondString = scnr.nextLine();
```

1) Hello there!
Welcome.

firstString gets _____ .

- Hello
- Hello there!
- Hello there! Welcome.

2) I
don't
know.

firstString gets _____ .

- I
- I don't
- I
  don't

3) Hey buddy.
What's up?

secondString gets _____ .

- Hey buddy.
- What's up?
- (empty)

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

4)

abc
def

secondString gets _____ . (Note that the first line above is blank).

- abc
- def
- (blank)

5) Walk away

firstString gets _____ . (Note the leading spaces before Walk).

- Walk away
- Walk away

Example: Getting multi-word names

Figure 1.8.3: Reading an input string containing spaces using nextLine.

```
import java.util.Scanner;

public class NameWelcome {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String firstName;
      String lastName;

      System.out.println("Enter first name: ");
      firstName = scnr.nextLine(); // Gets entire line
up to ENTER

      System.out.println("Enter last name: ");
      lastName = scnr.nextLine(); // Gets entire line
up to ENTER

      System.out.println();
      System.out.println("Welcome " + firstName + " " +
lastName + "!");
      System.out.println("May I call you " + firstName
+ "?");
   }
}
```

```
Enter first name:
Betty Sue
Enter last name:
McKay

Welcome Betty Sue
McKay!
May I call you
Betty Sue?
```


Mixing next() and nextLine()

Mixing next() and nextLine() can be tricky, because next() leaves the newline in the input, while nextLine() does not skip leading whitespace.

PARTICIPATION ACTIVITY 1.8.7: Combining next() and nextLine() can be tricky.

Animation captions:

1. Input characters include whitespace characters, normally invisible, but shown here using boxes labeled n (newline) and s (space).
2. str1 = scnr.next() will get "Kindness" into str1, stopping at the first whitespace, in this case a newline character.
3. The next statement, str2 = scnr.next(), will skip any leading whitespace (the newline, and the next line's leading spaces), then get the next characters "is" until the next whitespace.
4. Combining next() with nextLine() is a little tricky. The str1 = scnr.next() leaves the newline in the input. Then, str2 = nextLine() gets the rest of the line, which is blank.
5. When following next() with nextLine(), an extra nextLine() is needed to get past the newline left in the input by next().

PARTICIPATION ACTIVITY 1.8.8: Getting strings without and with whitespace.

Given the following input:

```
Every one
is great.
That's right.
```

1) What does the following get into str2?

```
str1 = scnr.next();
str2 = scnr.next();
```

- Every
- one
- Every one

2) What does the following get into str2?


```
str1 = scnr.next();
str2 = scnr.nextLine();
```

- Every one
- one
- one
  (has a leading space)

3) What does the following get into str2?

```
str0 = scnr.next();
str1 = scnr.next();
str2 = scnr.nextLine();
```

- one
- (blank)
- is great.

4) What does the following get into str2?

```
str0 = scnr.next();
str1 = scnr.next();
tmpStr = scnr.nextLine();
str2 = scnr.nextLine();
```

- (blank)
- is great.
- That's right.

5) What does the following get into str3?

```
str0 = scnr.next();
str1 = scnr.next();
tmpStr = scnr.nextLine();
str2 = scnr.nextLine();
str3 = scnr.nextLine();
```

- That's right.
- (blank)

6) What does the following put into str2?

```
str0 = scnr.next();
str1 = scnr.next();
tmpStr = scnr.next();
str2 = scnr.nextLine();
```

- is
- is great.
- great.
  (has a leading space)

CHALLENGE ACTIVITY 1.8.1: Representing text.

422352.2723990.qx3zqy7

Start

Type the program's output

```
public class StringOutput {
   public static void main(String[] args) {
      System.out.println("X");
   }
}
```

X

1 | 2 | 3 | 4 | 5

Check Next

CHALLENGE ACTIVITY 1.8.2: Reading and outputting strings.

Write a program that reads a person's first and last names separated by a space, assuming the first and last names are both single words. Then the program outputs last name, first name. End with newline.

Example output if the input is: Maya Jones

```
Jones, Maya
```

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class SpaceReplace {
4     public static void main(String[] args) {
5        Scanner scnr = new Scanner(System.in);
6        String firstName;
7        String lastName;
8
9        /* Your solution goes here  */
10
11    }
12 }
```


Run

View your last submission

1.9 Detecting ranges using logical operators

Logical AND, OR, and NOT (general)

A ***logical operator*** treats operands as being true or false, and evaluates to true or false. Logical operators include AND, OR, and NOT. Programming languages typically use various symbols for those operators, but below the words AND, OR, and NOT are used for introductory purposes.

PARTICIPATION ACTIVITY 1.9.1: Logical operators: AND, OR, and NOT.

Animation content:

Three truth tables are shown, each evaluating a different expression.

the first truth table takes in operands a and b and then uses the AND operator. False AND false evaluates to false. False AND true evaluates to false. True AND false evaluates to false. True AND true evaluates to true.

An example below is shown where x=7 and y=9. The expression reads (x > 0) AND (y < 10). Both operands are true so the entire expression evaluates to true. The next expression is (x > 0) AND (y < 5). The first operand is true, but the second is false, so the entire expression evaluates to false.

The second truth table takes in operands a and b and then uses the OR operator. False OR false evaluates to false. False OR true evaluates to true. True OR false evaluates to true. True OR true evaluates to true.

An example below is shown where x=7 and y=9. The expression reads (x < 0) OR (y > 10). Both operands are false so the entire expression evaluates to false. The next expression is (x < 0) AND (y > 5). The first operand is false, but the second is true, so the entire

expression evaluates to true.

The third truth table takes in operand a and then uses the NOT operator. False evaluates to true. True evaluates to false.

An example below is shown where x=7 and y=9. The expression reads NOT(x < 0). The operand alone evaluates to true, but the expression as a whole evaluates to false. The expression reads NOT(x > 0). The operand alone evaluates to false, but the expression as a whole evaluates to true.

Animation captions:

1. AND evaluates to true only if BOTH operands are true.
2. OR evaluates to true if ANY operand is true (one, the other, or both).
3. NOT evaluates to the opposite of the operand.
4. Each operand is commonly an expression itself. If x = 7, y = 9, then (x > 0) AND (y < 10) is true AND true, so evaluates to true (both operands are true).

Table 1.9.1: Logical operators.

Logical operator	Description
a AND b	***Logical AND***: true when both of its operands are true.
a OR b	***Logical OR***: true when at least one of its two operands are true.
NOT a	***Logical NOT***: true when its one operand is false, and vice-versa.

PARTICIPATION ACTIVITY 1.9.2: Evaluating expressions with logical operators.

Indicate whether the expression evaluates to true or false.
x is 7, y is 9.

1) x > 5

- true
- false

2) (x > 5) AND (y < 20)

- true
- false

3) (x > 10) AND (y < 20)

- true
- false

4) (x > 10) OR (y < 20)

- true
- false

5) (x > 10) OR (y > 20)

- true
- false

6) NOT (x > 10)

- true
- false

7) NOT ((x > 5) AND (y < 20))

- true
- false

Detecting ranges with logical operators (general)

A common use of logical operators is to detect if a value is within a range.

PARTICIPATION ACTIVITY 1.9.3: Using AND to detect if a value is within a range.

Animation content:

```
At the top, a purple line represents the wanted range 10 < x < 15.
```

Underneath this purple line are the numbers 10, 11, 12, 13, 14, and 15. The purple line only exists between 11 and 14.

Further below there is a red line that starts at 11 and stretches to the right to represent the range (10 < x): all values greater than 10. Above the red line, there is a blue line that starts at 14 and stretches to the left to represent the range (x < 15): all values less than 15.

The AND operator appears and combines the two range expressions (10 < x AND x < 15) and a second purple line is formed from where the red and blue lines overlap. The second purple line only exists between 11 and 14 to represent the range (10 < x) and (x < 15). Note: 10 < x < 15 is not a valid expression in Java. 10 < x AND x < 15 is a correct way to identify the range.

Animation captions:

1. The range 10 < x < 15 means that x may be 11, 12, 13, 14.
2. Specifying that range in a program can be done using two < operators along with an AND operator. 10 < x defines the range 11 and higher.
3. x < 15 defines the range 14 and lower. ANDing yields the overlapping range. Only when x is 11, 12, 13, or 14 will both expressions be true.

PARTICIPATION ACTIVITY 1.9.4: Using AND to detect if a value is within a range.

1) Which approach uses a logical operator to detect if x is in the range 1 to 99.

   - 0 < x < 100
   - (0 < x) AND (x < 100)
   - (0 < x) AND (x > 100)

2) Which detects if x is in the range -4 to +4?

   - (x < -5) AND (x < 5)
   - (x > -5) OR (x < 5)
   - (x > -5) AND (x < 5)

3) Which detects if x is either less than -5, or greater than 10?

- (x < -5) AND (x > 10)
- (x < -5) OR (x > 10)

Logical operators

Special symbols are used to represent the AND, OR, and NOT logical operators. Logical operators are commonly used in expressions of if-else statements.

Table 1.9.2: Logical operators.

Logical operator	Description
a && b	***Logical AND*** (&&): true when both of its operands are true
a || b	***Logical OR*** (||): true when at least one of its two operands are true
!a	***Logical NOT*** (!): true when its one operand is false, and vice-versa.

PARTICIPATION ACTIVITY 1.9.5: Logical operators.

Match the symbol with the logical operator.

If unable to drag and drop, refresh the page.

|| **!** **&&** **!!**

AND

OR

NOT

No such operator

Reset

PARTICIPATION ACTIVITY 1.9.6: Logical operators: Complete the expressions to detect the desired range.

1) daysLogged is greater than 30 and less than 90

```
if ( (daysLogged > 30)
     (daysLogged < 90)
) {
   ...
}
```

Check Show answer

2) 0 < maxCars < 100

```
if ( (maxCars > 0)
(maxCars < 100) ) {
   ...
}
```

Check Show answer

3) numStores is between 10 and 20, inclusive.

```
if ( (numStores >= 10)
     (numStores <= 20)
) {
   ...
}
```

Check Show answer

4) notValid is either less than 15, or greater than 79.

```
if ( (notValid < 15)
     (notValid > 79) )
{
   ...
}
```

PARTICIPATION ACTIVITY 1.9.7: Creating expressions with logical operators.

1) numDogs has a minimum of 2 and a maximum of 5.

```
if ( (numDogs >= 2)
                    ) {
   ...
}
```

Check Show answer

2) wage is greater than 10 and less than 18. Use > and < (not >= and <=). Use parentheses around sub-expressions.

```
if (
                    ) {
   ...
}
```

Check Show answer

3) num is a 3-digit positive integer. Ex: 100, 989, and 523, are 3-digit positive integers, but 55, 1000, and -4 are not.

For most direct readability, your expression should compare directly with the smallest and largest 3-digit number.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
if ( (num >= 100)
     
) {
   ...
}
```


A common error is to use & instead of && and | instead ||. & and | are not logical operators and may produce unexpected output.

PARTICIPATION ACTIVITY 1.9.8: Logical expression simulator.

Try typing different expressions involving x, y and observe whether the expression evaluates to true for different values of x, y.
Ex: Test the expression (x < 10) && (x >2) when x is assigned with 1, 2, 5, and 20.

```
int x = 7 ;
int y = 5 ;
if (                                ) {
   ...
}
```

Run code

Output is:

```
Awaiting your input...
```

Example: TV channels

A cable TV provider may have regular channels numbered 2-499, and high-definition channels numbered 1002-1499. A program may set a character variable to 's' for standard, 'h' for high-definition, and 'e' for error.

Figure 1.9.1: Detecting ranges: Cable TV channels.

```
if ((userChannel >= 2) && (userChannel <= 499)) {
   channelType = 's';
}
else if ((userChannel >= 1002) && (userChannel <=
1499)) {
   channelType = 'h';
}
else {
   channelType = 'e';
}
```


zyDE 1.9.1: Detecting ranges: Cable TV channels.

Run the program and observe the output. Change the input box value from 3 to anoth number, and run again.

Load default template...

```
1  import java.util.Scanner;
2
3  public class CableTVChannels {
4
5     public static void main(String[] args) {
6        int userChannel;
7        char channelType;
8        Scanner scnr = new Scanner(System.in);
9
10       userChannel = scnr.nextInt();
11
12       if ((userChannel >= 2) && (userChanne
13          channelType = 's';
14       }
15       else if ((userChannel >= 1002) && (us
16          channelType = 'h';
17       }
```

3

Run

PARTICIPATION ACTIVITY 1.9.9: TV channel example: Detecting ranges.

Consider the above example.

1) If userChannel is 300, to what does the if statement's expression,

`(userChannel >= 2) && (userChannel <= 499)`, evaluate?

- true
- false

2) If userChannel is 300, does the else if's expression `(userChannel >= 1002) && (userChannel <= 1499)` get checked?

- Yes
- No

3) Did the expressions use logical AND or logical OR?

- AND
- OR

4) Channels 500-599 are paid channels. Does this expression detect that range? `(userChannel >= 500) || (userChannel <= 599)`

- Yes
- No

Detecting ranges implicitly vs. explicitly

A programmer often uses logical operators to detect a range by explicitly specifying the high-end and low-end of the range. However, if a program should detect increasing ranges without gaps, a multi-branch if-else statement can be used without logical operators; the low-end of the range is implicitly known upon reaching an expression. Likewise, a decreasing range without gaps has implicitly-known high-ends.

PARTICIPATION ACTIVITY 1.9.10: Detecting ranges implicitly vs. explicitly.

Animation content:

On the left side is an example of explicitly defined ranges. The example reads as follows, with comments defining the ranges:

```
if (x < 0) {
```

```
 // Negative
}
else if ( (x >= 0) && (x <= 10) ) {
 // 0..10
}
else if ( (x >= 11) && (x <= 20) ) {
 // 11..20
}
else {
 // 21+
}
```


On the right side is an example of implicitly defined ranges. The example reads as follows, with comments defining the ranges:

```
if (x < 0) {
 // Negative
}
else if (x <= 10) {
 // 0..10
}
else if (x <= 20) {
 // 11..20
}
else {
 // 21+
}
```

In this example x < 0, x <= 10, and x <= 20 are all implicitly defining their respective ranges.

Animation captions:

1. This code detects ranges explicitly using the AND operator. The first branch executes when x < 0, the second when (x >=0) && (x <= 10).
2. But, if the first branch doesn't execute, x must be >= 0. So the second branch's expression can just be x <= 10. The x >= 0 is implicit.
3. Implicit ranges can simplify a multi-branch if statement for ranges without gaps.

PARTICIPATION ACTIVITY 1.9.11: Detecting ranges implicitly vs explicitly.

For each pair of statements, does the second if-else statement detect the same ranges as the first if-else statement?

1)

```
if (temp <= 0)...
else if ((temp > 0) && (temp <
100))...
```

```
if (temp <= 0)...
else if (temp < 100)...
```

- Yes
- No

2)

```
if (systolic < 130)...
else if ((systolic >= 130) &&
(systolic <= 139))...
```

```
if (systolic < 130)...
else if (systolic >= 130)...
```

- Yes
- No

3)

```
if ( (year >= 1901) && (year
<= 2000) )...
else if ((year >= 2001) &&
(year <= 2100))...
```

```
if (year <= 2000)...
else if (year <= 2100)...
```

- Yes
- No

CHALLENGE ACTIVITY 1.9.1: Detect number range.

Write an expression that prints "Eligible" if userAge is between 18 and 25 inclusive.
Ex: 17 prints "Ineligible", 18 prints "Eligible".

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class AgeChecker {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        int userAge;
7
8        userAge = scnr.nextInt();
9
10       if(/* Your solution goes here  */){
11           System.out.println("Eligible");
12       }
13       else{
14           System.out.println("Ineligible");
15       }
16    }
17 }
```


Run

View your last submission

1.10 Switch statements

Switch statement

A ***switch*** statement can more clearly represent multi-branch behavior involving a variable being compared to constant values. The program executes the first ***case*** whose constant expression matches the value of the switch expression, executes that case's statements, and then jumps to the end. If no case matches, then the ***default case*** statements are executed.

PARTICIPATION ACTIVITY 1.10.1: Switch statement.

Animation content:

A code snippet with comments reads as follows:

```
switch (a) {
  case 0:
     // Print "zero"
     break;
```

```
  case 1:
     // Print "one"
     break;

  case 2:
     // Print "two"
     break;

   default:
     // Print "unknown"
     break;
}
```

To the right of this are two cases with each having a separate output on a screen. In the first, variable a equals 1. The screen displays "one." In the second, variable a equals 5. The screen displays "unknown."

Animation captions:

1. A switch statement can more clearly represent multi-branch behavior involving a variable being compared to constant values.
2. The program executes the first case whose constant expression matches the value of the switch expression, executes that case's statements, and then jumps to the end.
3. If no case matches, then the default case statements are executed.

PARTICIPATION ACTIVITY 1.10.2: Switch statement.

numItems and userVal are int types. What is the final value of numItems for each userVal?

```
switch (userVal) {
   case 1:
      numItems = 5;
      break;

   case 3:
      numItems = 12;
      break;

   case 4:
      numItems = 99;
      break;

   default:
      numItems = 55;
      break;
}
```


1) userVal = 3;

Check Show answer

2) userVal = 0;

Check Show answer

3) userVal = 2;

Check Show answer

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Multi-branch if-else statement

A switch statement can be written using a multi-branch if-else statement, but the switch statement may make the programmer's intent clearer.

```
if (dogYears == 0) {           // Like case 0
   // Print 0..14 years
}
else if (dogYears == 1) {      // Like case 1
   // Print 15 years
}
...
else if (dogYears == 5) {      // Like case 5
   // Print 37 years
}
else {                         // Like default case
   // Print unknown
}
```


Switch statement general form

The switch statement's expression should be an integer, char, or string (discussed elsewhere). The expression should not be a Boolean or a floating-point type. Each case must have a constant expression like 2 or 'q'; a case expression cannot be a variable.

The order of cases doesn't matter assuming break statements exist at the end of each case. The earlier program could have been written with case 3 first, then case 2, then case 0, then case 1, for example (though that would be bad style).

Good practice is to always have a default case for a switch statement. A programmer may be sure all cases are covered only to be surprised that some case was missing.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Construct 1.10.1: Switch statement general form.

```
switch (expression) {
   case constantExpr1:
      // Statements
      break;

   case constantExpr2:
      // Statements
      break;

   ...

   default: // If no other case
matches
      // Statements
      break;
}
```


©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.10.1: Switch example: Estimates a dog's age in human years.

```
import java.util.Scanner;

/* Estimates dog's age in equivalent human years.
   Source: www.dogyears.com
*/

public class DogYears {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int dogAgeYears;

      System.out.print("Enter dog's age (in years):
");
      dogAgeYears = scnr.nextInt();

      switch (dogAgeYears) {
         case 0:
            System.out.println("That's 0..14 human
years.");
            break;

         case 1:
            System.out.println("That's 15 human
years.");
            break;

         case 2:
            System.out.println("That's 24 human
years.");
            break;

         case 3:
            System.out.println("That's 28 human
years.");
            break;

         case 4:
            System.out.println("That's 32 human
years.");
            break;

         case 5:
            System.out.println("That's 37 human
years.");
            break;

         default:
            System.out.println("Human years
unknown.");
            break;
      }
   }
}
```

```
Enter dog's age (in
years): 4
That's 32 human years.

...

Enter dog's age (in
years): 17
Human years unknown.
```


©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

zyDE 1.10.1: Switch statement: Numbers to words.

Extend the program for dogYears to support age of 6 to 10 years. Conversions are 6:
8:52, 9:57, 10:62.

Load default template...

```
1  import java.util.Scanner;
2
3  /* Estimates dog's age in equivalent human
4     Source: www.dogyears.com
5  */
6
7  public class DogYears {
8     public static void main(String[] args)
9        Scanner scnr = new Scanner(System.in
10       int dogAgeYears;
11
12       System.out.println("Enter dog's age
13       dogAgeYears = scnr.nextInt();
14
15       switch (dogAgeYears) {
16          case 0:
17             System.out.println("That's 0..
```

7

Run

Omitting the break statement

Omitting the ***break*** statement for a case will cause the statements within the next case to be executed. Such "falling through" to the next case can be useful when multiple cases, such as cases 0, 1, and 2, should execute the same statements.

The following extends the previous program for dog ages less than 1 year old. If the dog's age is 0, the program asks for the dog's age in months. Within the `switch (dogAgeMonths)` statement, "falling through" is used to execute the same display statement for several values of dogAgeMonths. For example, if dogAgeMonths is 0, 1 or 2, the same statement executes.

A common error occurs when the programmer forgets to include a break statement at the end of a case's statements.

Figure 1.10.2: Switch example: Dog years with months.

```
import java.util.Scanner;

public class DogYearsMonths {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int dogAgeYears;
      int dogAgeMonths;

      System.out.print("Enter dog's age (in years):
");
      dogAgeYears = scnr.nextInt();

      if (dogAgeYears == 0) {
         System.out.print("Enter dog's age in
months: ");
         dogAgeMonths = scnr.nextInt();

         switch (dogAgeMonths) {
            case 0:
            case 1:
            case 2:
               System.out.println("That's 0..14
human months.");
               break;

            case 3:
            case 4:
            case 5:
            case 6:
               System.out.println("That's 14 months
to 5 human years.");
               break;

            case 7:
            case 8:
               System.out.println("That's 5..9 human
years.");
               break;

            case 9:
            case 10:
            case 11:
            case 12:
               System.out.println("That's 9..15
human years.");
               break;

            default:
               System.out.println("Invalid input.");
               break;
         }
      }
      else {
```

```
Enter dog's age (in
years): 0
Enter dog's age in
months: 7
That's 5..9 human
years.

...

Enter dog's age (in
years): 4
FIXME: Do earlier dog
years cases
```

```
            System.out.println("FIXME: Do earlier dog
 years cases");
            switch (dogAgeYears) {
            }
        }
    }
}
```


PARTICIPATION ACTIVITY 1.10.3: Switch statement.

userChar is a char and encodedVal is an int. What will encodedVal be for each userChar value?

```
switch (userChar) {
   case 'A':
      encodedVal = 1;
      break;

   case 'B':
      encodedVal = 2;
      break;

   case 'C':

   case 'D':
      encodedVal = 4;
      break;

   case 'E':
      encodedVal = 5;

   case 'F':
      encodedVal = 6;
      break;

   default:
      encodedVal = -1;
      break;
}
```

1) userChar = 'A'

Check Show answer

2) userChar = 'B'

Check Show answer

3) userChar = 'C'

Check Show answer

4) userChar = 'E'

Check Show answer

5) userChar = 'G'

Check Show answer

CHALLENGE ACTIVITY 1.10.1: Rock-paper-scissors.

Write a switch statement that checks nextChoice. If 0, print "Rock". If 1, print "Paper". If 2, print "Scissors". For any other value, print "Unknown". End with newline.

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class Roshambo {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        int nextChoice;
7
8        nextChoice = scnr.nextInt();
9
10       /* Your solution goes here  */
11
12    }
13 }
```

Run

View your last submission

CHALLENGE ACTIVITY | 1.10.2: Switch statement to convert letters to Greek letters.

Write a switch statement that checks origLetter. If 'a' or 'A', print "Alpha". If 'b' or 'B', print "Beta". For any other character, print "Unknown". Use fall-through as appropriate. End with newline.

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class ConvertToGreek {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        char origLetter;
7
8        origLetter = scnr.next().charAt(0);
9
10       /* Your solution goes here  */
11
12    }
13 }
```

Run

View your last submission

1.11 String comparisons

String comparison: Equality

Two strings are commonly compared for equality. Equal strings have the same number of characters, and each corresponding character is identical.

PARTICIPATION ACTIVITY 1.11.1: Equal strings.

Which strings are equal?

1) "Apple", "Apple"
   - Equal
   - Unequal

2) "Apple", "Apples"
   - Equal
   - Unequal

3) "Apple pie!!", "Apple pie!!"
   - Equal
   - Unequal

4) "Apple", "apple"
   - Equal
   - Unequal

A programmer can compare two strings using the notation `str1.equals(str2)`. The ***equals*** method returns true if the two strings are equal. A common error is to use == to compare two strings, which behaves differently than expected.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.11.1: String equality example.

```
import java.util.Scanner;

public class StringEquality {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String userWord;

      System.out.print("Enter a word: ");
      userWord = scnr.next();

      if (userWord.equals("USA")) {
         System.out.println("United States of
America");
      }
      else {
         System.out.println(userWord);
      }
   }
}
```

```
Enter a word: Sally
Sally

...

Enter a word: USA
United States of
America

...

Enter a word: usa
usa
```


PARTICIPATION ACTIVITY 1.11.2: Comparing strings for equality.

To what does each expression evaluate? Assume str1 is "Apples" and str2 is "apples".

1) str1.equals("Apples")

- True
- False

2) str1.equals(str2)

- True
- False

3) !str2.equals("oranges")

- True
- False

4) A good way to compare strings is:
str1 == str2.

- True

String comparison: Relational

Strings are sometimes compared relationally (less than, greater than), as when sorting words alphabetically. A comparison begins at index 0 and compares each character until the evaluation results in false, or the end of a string is reached. 'A' is 65, 'B' is 66, etc., while 'a' is 97, 'b' is 98, etc. So "Apples" is less than "apples" because 65 is less than 97.

PARTICIPATION ACTIVITY 1.11.3: String comparison.

Animation captions:

1. A string comparison compares each character. The first five characters of studentName and teacherName are the same.
2. 'J' is greater than 'A', so studentName is greater than teacherName.

PARTICIPATION ACTIVITY 1.11.4: Case matters in string comparisons.

Indicate the result of comparing the first string with the second string.

1) "Apples", "Oranges"
   - less than
   - equal
   - greater than

2) "merry", "Merry"
   - less than
   - equal
   - greater than

3) "banana", "bananarama"
   - less than
   - equal
   - greater than

A programmer compares strings relationally using the notation str1.compareTo(str2). ***compareTo()***

returns values as follows.

Table 1.11.1: str1.compareTo(str2) return values.

Relation	Returns	Expression to detect
str1 less than str2	Negative number	str1.compareTo(str2) < 0
str1 equal to str2	0	str1.compareTo(str2) == 0
str1 greater than str2	Positive number	str1.compareTo(str2) > 0

A common error is to forget that case matters in a string comparison. A programmer can compare strings while ignoring case using str1.***equalsIgnoreCase***(str2) and str1.***compareToIgnoreCase***(str2).

PARTICIPATION ACTIVITY 1.11.5: Relational string comparison.

1) Write an expression that evaluates to true if myName is greater than yourName.

```
if
(myName.compareTo(yourName)
        0) {
   ...
}
```

Check Show answer

2) Write an expression that evaluates to true if authorName1 is less than authorName2.

```
if (

) {
   ...
}
```

Check Show answer

CHALLENGE ACTIVITY 1.11.1: String comparison: Detect word.

Write an if-else statement that prints "Goodbye" if userString is "Quit", else prints "Hello". End with newline.

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class DetectWord {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        String userString;
7
8        userString = scnr.next();
9
10       /* Your solution goes here  */
11
12    }
13 }
```

Run

View your last submission

CHALLENGE ACTIVITY 1.11.2: Print two strings in alphabetical order.

Print the two strings, firstString and secondString, in alphabetical order. Assume the strings are lowercase. End with newline. Sample output:

```
capes rabbits
```


422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class OrderStrings {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
```

Run

View your last submission

1.12 For loops

Survey

The following questions are part of a zyBooks survey to help us improve our content so we can offer the best experience for students. The survey can be taken anonymously and takes just 3-5 minutes. Please take a short moment to answer by clicking the following link.

Link: Student survey

Basics

A loop commonly must iterate a specific number of times, such as 10 times. Though achievable with a while loop, that situation is so common that a special kind of loop exists. A ***for loop*** is a loop with three parts at the top: a loop variable initialization, a loop expression, and a loop variable update. A for loop describes iterating a specific number of times more naturally than a while loop.

Construct 1.12.1: For loop.

```
for (initialExpression; conditionExpression;
updateExpression) {
  // Loop body
}
// Statements after the loop
```


PARTICIPATION ACTIVITY 1.12.1: For loops.

Animation content:

A code snippet to the left reads:

```
int i;

i = 0;
while (i < 5) {
    // Loop body
    i = i + 1;
}
```

A code snippet to the right reads:

```
int i;

for (i = 0; i < 5; i = i + 1) {
    // Loop body
}
```


Underneath is a walkthrough of the code for each value of the variable i.

At i = 0, 1, 2, 3, and 4, the loop iterates (5 iterations). At i = 5, the loop does not iterate.

Animation captions:

1. This while loop pattern with i = 0 before, loop expression i < 5, and loop body ending with i = i + 1, iterates 5 times: when i = 0, 1, 2, 3, and 4.
2. The pattern is so common that a special construct, a for loop, exists to collect the three parts in one place at the loop's top, improving readability and reducing errors.
3. Note that semicolons separate the three parts. No semicolon is needed at the end.

The statement i = i + 1 is so common that the language supports the shorthand ***++i***, with ++ known as the ***increment operator***. (Likewise, **--** is the ***decrement operator***, --i means i = i - 1). As such, a standard way to loop N times is shown below.

Figure 1.12.1: A standard way to loop N times, using a for loop.

```
int i;
...
for (i = 0; i < N; ++i)
{
   ...
}
```

PARTICIPATION ACTIVITY 1.12.2: For loops.

1) What are the values of i for each iteration of:

```
for (i = 0; i < 6; ++i) {
   ...
}
```

- 1, 2, 3, 4, 5
- 0, 1, 2, 3, 4, 5
- 0, 1, 2, 3, 4, 5, 6

2) How many times will this loop iterate?

```
for (i = 0; i < 8; ++i) {
   ...
}
```


- 7 times
- 8 times
- 9 times

3) Goal: Loop 10 times

```
for (i = 0; _____; ++i) {
   ...
}
```

- i < 9
- i < 10
- i < 11

4) Goal: Loop 99 times

```
for (i = 0; _____; ++i) {
   ...
}
```

- i < 99
- i <= 99
- i == 99

5) Goal: Loop 20 times

```
for (____; i < 20; ++i) {
   ...
}
```

- i = 0
- i = 1

6) Goal: Loop numYears times (numYears is an int variable).

```
for (i = 0; _____; ++i) {
   ...
}
```

- numYears
- i <= numYears
- i < numYears

PARTICIPATION ACTIVITY 1.12.3: For loops.

Write for loops using the following form:

```
for (i = 0; i < 10; ++i) {
```

Note: Using a variable other than i is not accepted by this activity.

1) Complete the for loop to iterate 5 times. (Don't forget the

semicolon).

```
for (i = 0;
    [          ] ++i)
{
   ...
}
```

Check Show answer

2) Complete the for loop to iterate 7 times.

```
for (
    [          ] ++i)
{
   ...
}
```

Check Show answer

3) Complete the for loop to iterate 500 times. (Don't forget the parentheses).

```
for [          ]
{
   ...
}
```

Check Show answer

4) Complete the for loop to iterate numDogs times. numDogs is an int variable.

```
for (i = 0;
    [          ] ++i)
{
   ...
}
```

Check Show answer

*Note: Actually two increment operators exist: ++i (**pre-increment**) and i++ (**post-increment**). ++i increments before evaluating to a value, while i++ increments after. Ex: If i is 5, outputting ++i outputs 6, while outputting i++ outputs 5 (and then i becomes 6). This material primarily uses ++i for simplicity and safety, although many programmers use i++, especially in for loops.*

Example: Savings with interest

The following program outputs the amount of a savings account each year for 10 years, given an input initial amount and interest rate. A for loop iterates 10 times, such that each iteration represents one year, outputting that year's savings amount.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.12.2: For loop: Savings interest program.

```
import java.util.Scanner;

public class SavingsInterestCalc {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      double initialSavings;  // User-entered
initial savings
      double interestRate;    // Interest rate
      double currSavings;     // Current savings
with interest
      int i;                  // Loop variable

      System.out.print("Enter initial savings: ");
      initialSavings = scnr.nextDouble();

      System.out.print("Enter interest rate: ");
      interestRate = scnr.nextDouble();

      System.out.println("\nAnnual savings for 10
years: ");

      currSavings = initialSavings;
      for (i = 0; i < 10; ++i) {
         System.out.println("$" + currSavings);
         currSavings = currSavings + (currSavings
* interestRate);
      }
   }
}
```



```
Enter initial savings:
10000
Enter interest rate:
0.05

Annual savings for 10
years:
$10000.0
$10500.0
$11025.0
$11576.25
$12155.0625
$12762.815625
$13400.95640625
$14071.0042265625
$14774.554437890625
$15513.282159785156
```

PARTICIPATION ACTIVITY 1.12.4: Savings interest program.

Consider the example above.

1) How many times does the for loop iterate?

   - 5
   - 10

2) During each iteration, the loop body's statements output the current savings amount, and then _____.

- increment i
- update currSavings

3) Can the input values change the number of loop iterations?

- Yes
- No

Example: Computing the average of a list of input values

The example below computes the average of an input list of integer values. The first input indicates the number of values in the subsequent list. That number controls how many times the subsequent for loop iterates.

Figure 1.12.3: Computing an average, with first value indicating list size.

```
import java.util.Scanner;

public class ListAverage {
   public static void main(String [] args) {
      Scanner scnr = new Scanner(System.in);
      int currValue;
      int valuesSum;
      int numValues;
      int i;

      numValues = scnr.nextInt(); // Gets number of values
in list

      valuesSum = 0;

      for (i = 0; i < numValues; ++i) {
         currValue = scnr.nextInt(); // Gets next value in
list
         valuesSum += currValue;
      }

      System.out.println("Average: " + (valuesSum /
numValues));
   }
}
```

```
4 10 1 6 3
Average: 5

...

5 -75 -50 30 60
80
Average: 9
```


PARTICIPATION ACTIVITY 1.12.5: Computing the average.

Consider the example above, with input 4 10 1 6 3. Note: The first input indicates the

number of values in the subsequent list.

1) Before the loop is entered, what is valuesSum?

Check Show answer

2) What is valuesSum after the first iteration?

Check Show answer

3) What is valuesSum after the second iteration?

Check Show answer

4) valuesSum is 20 after the fourth iteration. What is numValues?

Check Show answer

5) For the following input, how many times will the for loop iterate?
7 -1 -3 -5 -14 -15 -20 -40

Check Show answer

Choosing among for and while loops

Generally, a programmer uses a for loop when the number of iterations is known (like loop 5 times, or loop numItems times), and a while loop otherwise.

Table 1.12.1: Choosing between while and for loops: General guidelines (not strict rules though).

for	Number of iterations is computable before the loop, like iterating N times.
while	Number of iterations is not (easily) computable before the loop, like iterating until the input is 'q'.

PARTICIPATION ACTIVITY 1.12.6: While loops and for loops.

Choose the most appropriate loop type.

1) Iterate as long as user-entered char c is not 'q'.
   - ○ while
   - ○ for

2) Iterate until the values of x and y are equal, where x and y are changed in the loop body.
   - ○ while
   - ○ for

3) Iterate 100 times.
   - ○ while
   - ○ for

CHALLENGE ACTIVITY 1.12.1: Enter the for loop's output.

422352.2723990.qx3zqy7

Start

Type the program's output

```
public class ForLoopOutput {
   public static void main (String [] args) {
      int i;

      for (i = 0; i < 4; ++i) {
         System.out.print(i);
      }
   }
}
```

0123

1 2

Check Next

1.13 User-defined method basics

Methods (general)

A program may perform the same operation repeatedly, causing a large and confusing program due to redundancy. Program redundancy can be reduced by creating a grouping of predefined statements for repeatedly used operations, known as a ***method***. Even without redundancy, methods can prevent a main program from becoming large and confusing.

PARTICIPATION ACTIVITY | 1.13.1: Methods can reduce redundancy and keep the main program simple.

Animation content:

```
In the top left there is a blue box containing the following code:
Main program
    c1 = (f1 - 32.0) * (5.0 / 9.0)
    c2 = (f2 - 32.0) * (5.0 / 9.0)
    c3 = (f3 + 32.0) * (5.0 / 9.0)

The text "Cluttered Repeated code" is on the right of the blue box.
The text "Oops, error in third calculation." is underneath the last
line of code in the blue box.

In the top right there are two blue boxes.
The first blue box contains the following code:
F2C(f)
```

```
    c = (f - 32.0) * (5.0 / 9.0)
    return c
```

The first blue box has the text "Calculation only written once" next to it.

The second blue box contains the following code:

```
Main program
    c1 = F2C(f1)
    c2 = F2C(f2)
    c3 = F2C(f3)
```


The second blue box has the text "Simpler" next to it.

In the bottom left are two versions of the same code.
The first version of the code is written in gray text. The code is the following:

```
Main program
    a = expression1
    b = a + expression2
    c = b * expression3

    d = expression1
    e = d + expression2
    f = e * expression3
```

The lines that define a, b, and c and the lines that define d, e, and f are labelled "XYZ".

The second version of the code is written in black text. The code is the following:

```
Main program
    c = XYZ(a, b)
    f = XYZ(d, e)
```


In the bottom right are two versions of the same code.
The first version of the code is written in gray text. The code is the following:

```
Main program
```

```
    a = expression1
    b = a + expression2
    c = b * expression3

    d = expression4
    e = d * d * d
    f = (e + 1) * 2
    f = CalcPQR(d, e, g)
```


The lines that define a, b, and c are labelled "XYZ". THe lines that define d, e, and f are labelled "PQR".

The second version of the code is written in black text. The code is the following:

```
Main program
    c = XYZ(a, b)
    f = CalcPQR(d, e, g)
```

Animation captions:

1. Commonly, a program performs the same operation, such as a calculation, in multiple places. Here, the Fahrenheit to Celsius calculation is done in three places.
2. Repeated operations clutter the main program. And such repeated operations are more prone to errors.
3. A better approach defines the Fahrenheit to Celsius calculation once, named F2C here. Then, F2C can be "called" three times, yielding a simpler main program.
4. The impact is even greater when the operation has multiple statements -- here 3 statements, but commonly tens of statements. The main program is much simpler.
5. Even without repeated operations, calling predefined operations keeps the main program simple and intuitive.

PARTICIPATION ACTIVITY 1.13.2: Reasons for methods.

Consider the animation above.

1) In the original main program, the Fahrenheit to Celsius calculation appeared how many times?

2) Along with yielding a simpler main program, using the predefined Fahrenheit to Celsius calculation prevented what error in the original program?

- 1
- 3
- Adding rather than subtracting 32.0
- Multiplying by 9.0 / 5.0 rather than by 5.0 / 9.0

3) In the last example above, the main program was simplified by _____.

- eliminating redundant code for operation XYZ
- predefining operations for XYZ and CalcPQR

Basics of methods

A ***method*** is a named list of statements.

- A ***method definition*** consists of the new method's name and a block of statements. Ex: `public static double calcPizzaArea() { /* block of statements */ }`
- A ***method call*** is an invocation of a method's name, causing the method's statements to execute.

The method's name can be any valid identifier. A ***block*** is a list of statements surrounded by braces.

Below, the method call calcPizzaArea() causes execution to jump to the method's statements. Execution returns to the original location after executing the method's last statement.

PARTICIPATION ACTIVITY 1.13.3: Method example: Printing a pizza area.

Animation content:

```
On the left is the following code in a blue box:
public class PizzaArea {
   public static double calcPizzaArea() {
      double pizzaDiameter;
      double pizzaRadius;
```

```
      double pizzaArea;
      double piVal = 3.14159265;

      pizzaDiameter = 12.0;
      pizzaRadius = pizzaDiameter / 2.0;
      pizzaArea = piVal * pizzaRadius * pizzaRadius;
      return pizzaArea;
   }

   public static void main(String[] args) {
      System.out.println("12 inch pizza is " +
       calcPizzaArea() + " inches squared");
   }
}
```


On the right is a box containing the console output. The console output is the following:

```
12 inch pizza is 113.097 inches squared.
```

Animation captions:

1. The method call calcPizzaArea() jumps execution to the method's statements.
2. After the last statement of the calcPizzaArea() method, execution returns to the original location and the area of the pizza is returned and printed in main().

Methods must be defined within a class. The line: `public class PizzaArea {` begins a new class. The class has two methods, calcPizzaArea() and main(). Both use the ***access modifiers*** `public static`. public indicates the method may be called from any class in the program, and static indicates the method only uses values that are passed to the method; details of access modifiers are discussed elsewhere. For now, just know that a method defined using public static can be called from the program's main() method.

PARTICIPATION ACTIVITY 1.13.4: Method basics.

Given the calcPizzaArea() method defined above, and the following main() method:

```
public static void main (String [] args) {
   System.out.println("12 inch pizza is " + calcPizzaArea() + " square
inches");
   System.out.println("12 inch pizza is " + calcPizzaArea() + " square
inches");
}
```

1) How many method calls to calcPizzaArea() exist in main()?

Check Show answer

2) How many method definitions of calcPizzaArea() exist *within* main()?

Check Show answer

3) How many output statements exist in main()?

Check Show answer

4) How many output statements exist in calcPizzaArea()?

Check Show answer

5) Is main() itself a method? Answer yes or no.

Check Show answer

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Returning a value from a method

A method may return one value using a ***return statement***. Below, the computeSquare() method is defined to have a return type of int; thus, the method's return statement must have an expression that evaluates to an int.

PARTICIPATION ACTIVITY 1.13.5: Method returns computed square.

Animation content:

```
On the left is the following code in a blue box:
public class SquareComputation {

   public static int computeSquare(int numToSquare) {
      return numToSquare * numToSquare;
   }

   public static void main (String [] args) {
      int numSquared;

      numSquared = computeSquare(7);
      System.out.println("7 squared is " + numSquared);
  }
}

On the right is a box containing the console output. The console
output is the following:
7 squared is 49
```


Animation captions:

1. computeSquare() is called, passing 7 to the method's numToSquare parameter.
2. The method computes the square of the parameter numToSquare.
3. computeSquare(7) evaluates to 49, which is then assigned to numSquared.

Other return types are allowed, such as char, double, etc. A method can only return one item, not two or more. A return type of ***void*** indicates that a method does not return any value.

PARTICIPATION ACTIVITY 1.13.6: Return.

Given the definition below, indicate which are valid return statements:

```
int calculateSomeValue(int num1, int num2) { ... }
```

1) `return 9;`

- Yes
- No

2) `return num1;`

- Yes
- No

3) `return (num1 + num2) + 1;`

- Yes
- No

4) `return;`

- Yes
- No

5) `return num1 num2;`

- Yes
- No

6) `return (0);`

- Yes
- No

PARTICIPATION ACTIVITY 1.13.7: More on return.

1) The following is a valid method definition:
`char getItem() { ... }`

- True
- False

2) The following is a valid method definition for a method that returns two items:
`int, int getItems() { ... }`

- True

- False

3) The following is a valid method definition:

```
void printItem() { ... }
```

- True
- False

Parameters

A programmer can influence a method's behavior via an input.

- A ***parameter*** is a method input specified in a method definition. Ex: A pizza area method might have diameter as an input.
- An ***argument*** is a value provided to a method's parameter during a method call. Ex: A pizza area method might be called as calcPizzaArea(12.0) or as calcPizzaArea(16.0).

A parameter is like a variable declaration. Upon a call, the parameter's memory location is allocated, and the parameter is assigned with the argument's value. Upon returning to the original call location, the parameter is deleted from memory.

An argument may be an expression, like 12.0, x, or x * 1.5.

PARTICIPATION ACTIVITY 1.13.8: Method with parameters example: Calculating pizza area for different diameters.

Animation content:

On the top is the following code in a blue box:

```
public class MultiplePizzaAreas {

   public static double calcPizzaArea(double pizzaDiameter) {
      double pizzaRadius;
      double pizzaArea;
      double piVal = 3.14159265;

      pizzaRadius = pizzaDiameter / 2.0;
      pizzaArea = piVal * pizzaRadius * pizzaRadius;
      return pizzaArea;
   }
```

```
   public static void main (String [] args) {
      System.out.print("12.0 inch pizza is " +
         calcPizzaArea(12) + " inches squared");
      System.out.print("16.0 inch pizza is " +
         calcPizzaArea(16) + " inches squared");
   }
}
```



```
On the bottom is a box containing the console output. The console
output is the following:
12.0 inch pizza is 113.09733540000002 inches squared.
16.0 inch pizza is 201.0619296 inches squared.
```

Animation captions:

1. The method call calcPizzaArea() jumps execution to the method's statements, passing 12.0 to the method's pizzaDiameter parameter.
2. The next method call passes 16.0 to the method's pizzaDiameter parameter, which results in a different pizza area.

PARTICIPATION ACTIVITY 1.13.9: Parameters.

1) Complete the method beginning to have a parameter named numServings of type int.

```
int calcCalories(
                 ) {
```

Check Show answer

2) Write a statement that calls a method named calcCalories, passing the value 3 as an argument.

Check Show answer

3) Is the following a valid method definition beginning? Type yes or no.

```
int myMthd(int userNum
+ 5) { ... }
```

Check Show answer

4) Assume a method `int getBirthdayAge(int userAge)` simply returns the value of userAge + 1. What will the following output?

```
System.out.print(getBirthdayAge(42));
System.out.print(getBirthdayAge(20));
```

Check Show answer

Multiple or no parameters

A method definition may have multiple parameters, separated by commas. Parameters are assigned with argument values by position: First parameter with first argument, second with second, etc.

A method definition with no parameters must still have the parentheses, as in: `int doSomething() { ... }`. The call must include parentheses, with no argument, as in: `doSomething()`.

Figure 1.13.1: Method with multiple parameters.

```
public class PizzaVolume {

   public static double calcPizzaVolume(double pizzaDiameter, double
pizzaHeight) {
      double pizzaRadius;
      double pizzaArea;
      double pizzaVolume;
      double piVal = 3.14159265;

      pizzaRadius = pizzaDiameter / 2.0;
      pizzaArea = piVal * pizzaRadius * pizzaRadius;
      pizzaVolume = pizzaArea * pizzaHeight;
      return pizzaVolume;
   }

   public static void main (String [] args) {
      System.out.println("12.0 x 0.3 inch pizza is "  +
calcPizzaVolume(12.0, 0.3) + " inches cubed");
      System.out.println("12.0 x 0.8 inch pizza is "  +
calcPizzaVolume(12.0, 0.8) + " inches cubed");
      System.out.println("16.0 x 0.8 inch pizza is "  +
calcPizzaVolume(16.0, 0.8) + " inches cubed");
   }
}
```

```
12.0 x 0.3 inch pizza is 33.92920062 inches cubed.
12.0 x 0.8 inch pizza is 90.47786832000003 inches cubed.
16.0 x 0.8 inch pizza is 160.84954368 inches cubed.
```


PARTICIPATION ACTIVITY 1.13.10: Multiple parameters.

1) Which correctly defines two integer parameters x and y for a method definition:
`int calcVal(...)`?

   - (int x; int y)
   - (int x, y)
   - (int x, int y)

2) Which correctly passes two integer arguments for the method call:
`calcVal(...)`?

- (99, 44 + 5)
- (int 99, 44)
- (int 99, int 44)

3) Given a method definition:

```
int calcVal(int a, int b,
int c)
```

b is assigned with what value during this method call:

```
calcVal(42, 55, 77);
```

- Unknown
- 42
- 55

4) Given a method definition:

```
int calcVal(int a, int b,
int c)
```

and given int variables i, j, and k, which are valid arguments in the call `calcVal(...)`?

- (i, j)
- (k, i + j, 99)
- (i + j + k)

PARTICIPATION ACTIVITY 1.13.11: Calls with multiple parameters.

Given:

```
public static int getSum(int num1, int num2) {

   return num1 + num2;
}
```

1) What will be returned for the following method call?

```
getSum(1, 2);
```

Check Show answer

2) Write a statement that calls

getSum() to return the sum of x and 400 (providing the arguments in that order). End with ;

Check Show answer

Calling methods from methods

A method's statements may call other methods. In the example below, the pizzaCalories() method calls the calcCircleArea() method. (Note that main() itself is the first method called when a program executes, and calls other methods.)

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.13.2: Methods calling methods.

```
public class MethodsCallingMethods {

   public static double calcCircleArea(double circleDiameter) {
      double circleRadius;
      double circleArea;
      double piVal = 3.14159265;

      circleRadius = circleDiameter / 2.0;
      circleArea = piVal * circleRadius * circleRadius;

      return circleArea;
   }

   public static double pizzaCalories(double pizzaDiameter) {
      double totalCalories;
      double caloriesPerSquareInch = 16.7;    // Regular crust pepperoni
pizza

      totalCalories = calcCircleArea(pizzaDiameter) *
caloriesPerSquareInch;

      return totalCalories;
   }

   public static void main (String [] args) {
      System.out.printf("12 inch pizza has %.2f calories.\n",
pizzaCalories(12.0));
      System.out.printf("14 inch pizza has %.2f calories.\n",
pizzaCalories(14.0));
   }
}
```

```
12 inch pizza has 1888.73 calories.
14 inch pizza has 2570.77 calories.
```


Exploring further:

- Methods tutorial from Oracle's Java tutorials.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

CHALLENGE ACTIVITY 1.13.1: Basic method call.

Complete the method definition to return the hours given minutes. Output for sample program:

3.5

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2  public class HourToMinConv {
3     public static double getMinutesAsHours(double origMinutes) {
4
5        /* Your solution goes here  */
6
7     }
8
9     public static void main (String [] args) {
10       Scanner scnr = new Scanner(System.in);
11       double minutes;
12
13       minutes = scnr.nextDouble();
14
15       // Will be run with 210.0, 3600.0, and 0.0.
16       System.out.println(getMinutesAsHours(minutes));
17    }
```


Run

View your last submission

CHALLENGE ACTIVITY | 1.13.2: Enter the output of the returned value.

422352.2723990.qx3zqy7

Start

Type the program's output

```
public class ReturnMethodOutput {
   public static int changeValue(int x) {
      return x + 3;
   }

   public static void main (String [] args) {
      System.out.println(changeValue(2));
   }
}
```

5

1 | 2

Check Next

CHALLENGE ACTIVITY 1.13.3: Method definition: Volume of a pyramid.

Define a method pyramidVolume with double data type parameters baseLength, baseWidth, and pyramidHeight, that returns as a double the volume of a pyramid with a rectangular base. Relevant geometry equations:
Volume = base area x height x 1/3
Base area = base length x base width.
(Watch out for integer division).

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class CalcPyramidVolume {
4
5     /* Your solution goes here  */
6
7     public static void main (String [] args) {
8        Scanner scnr = new Scanner(System.in);
9        double userLength;
10       double userWidth;
11       double userHeight;
12
13       userLength = scnr.nextDouble();
14       userWidth = scnr.nextDouble();
15       userHeight = scnr.nextDouble();
16
17       System.out.println("Volume: " + pyramidVolume(userLength, userWidth, userHeight));
```

Run

View your last submission

1.14 Method name overloading

Sometimes a program has two methods with the same name but differing in the number or types of parameters, known as ***method name overloading*** or just ***method overloading***. The following two methods print a date given the day, month, and year. The first method has parameters of type int, int, and int, while the second has parameters of type int, string, and int.

Figure 1.14.1: Overloaded method name.

```
public class DatePrinter {
   public static void printDate(int currDay, int currMonth,
int currYear) {
      System.out.print(currMonth + "/" + currDay + "/" +
currYear);
   }

   public static void printDate(int currDay, String
currMonth, int currYear) {
      System.out.print(currMonth + " " + currDay + ", " +
currYear);
   }

   public static void main(String[] args) {

      printDate(30, 7, 2012);
      System.out.println();

      printDate(30, "July", 2012);
      System.out.println();
   }
}
```



```
7/30/2012
July 30,
2012
```

The compiler determines which method to call based on the argument types. printDate(30, 7, 2012) has argument types int, int, int, so calls the first method. printDate(30, "July", 2012) has argument types int, string, int, so calls the second method.

More than two same-named methods is allowed as long as each has distinct parameter types. Thus, in the above program:

- printDate(int month, int day, int year, int style) can be added because the types int, int, int, int differ from int, int, int, and from int, string, int.
- printDate(int month, int day, int year) yields a compiler error, because two methods have types int, int, int (the parameter names are irrelevant).

A method's return type does not influence overloading. Thus, having two same-named method definitions with the same parameter types but different return types still yield a compiler error.

PARTICIPATION ACTIVITY 1.14.1: Method name overloading.

Given the following method definitions, select the number that each method call would print. If the method call would not compile, choose Error.

```
public class DatePrinter {
   public static void printDate(int day, int month, int year) {
      System.out.println("1");
   }

   public static void printDate(int day, String month, int year) {
      System.out.println("2");
   }

   public static void printDate(int month, int day) {
      System.out.println("3");
   }
}
```


1) printDate(30, 7, 2012);

- 1
- 2
- 3
- Error

2) printDate(30, "July", 2012);

- 1
- 2
- 3
- Error

3) printDate(7, 2012);

- 1
- 2
- 3
- Error

4) printDate(30, 7);

- 1
- 2
- 3
- Error

5) printDate("July", 2012);

- 1
- 2

Exploring further:

- Method definition and overloading from Oracles' Java tutorials

CHALLENGE ACTIVITY 1.14.1: Overload salutation printing.

Complete the second printSalutation() method to print the following given personName "Holly" and customSalutation "Welcome":

```
Welcome, Holly
```

End with a newline.

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class MultipleSalutations {
4     public static void printSalutation(String personName) {
5        System.out.println("Hello, " + personName);
6     }
7
8     //Define void printSalutation(String personName, String customSalutation)...
9
10    /* Your solution goes here  */
11
12    public static void main (String [] args) {
13       printSalutation("Holly", "Welcome");
14       printSalutation("Sanjiv");
15    }
16 }
```

Run

View your last submission

CHALLENGE ACTIVITY 1.14.2: Convert a height into inches.

Write a second convertToInches() with two double parameters, numFeet and numInches, that returns the total number of inches. Ex: convertToInches(4.0, 6.0) returns 54.0 (from 4.0 * 12 + 6.0).

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class FunctionOverloadToInches {
4
5     public static double convertToInches(double numFeet) {
6        return numFeet * 12.0;
7     }
8
9     /* Your solution goes here  */
10
11    public static void main (String [] args) {
12       double totInches;
13
14       totInches = convertToInches(4.0, 6.0);
15       System.out.println("4.0, 6.0 yields " + totInches);
16
17       totInches = convertToInches(5.8);
```


Run

View your last submission

1.15 Using references in methods

Storage of primitive data types and arrays

Arrays can be used in methods just like int and double values. However, an array is stored in memory differently than variables of primitive data types, like int or double. An int and double variable is stored directly in the stack frame. An array is stored indirectly in the heap, and only the reference to the array is stored in the stack frame. This subtle difference in storage means that arrays behave differently in methods than int and double variables.

PARTICIPATION ACTIVITY 1.15.1: Primitive data and arrays are stored differently in memory.

Animation captions:

1. The int variable twoYearOlds is stored in a stack frame.
2. The array childrenAges is constructed and stored on the heap. The reference of the array is stored in the stack frame.

Passing an int variable as an argument to a method results in the parameter having a separate copy of the int variable's value. If this value is changed inside the method, the value of the argument will not be changed because primitive data types are passed by value. Ex: A method cannot swap two variables of primitive data types such as two int variables.

PARTICIPATION ACTIVITY 1.15.2: A method cannot swap two int values.

Animation captions:

1. The variables myVal and yourVal are stored in the stack frame.
2. When swapWRONG() method call copies the values 5 and 7 to the parameters firstVal and secondVal, which are stored in the swapWRONG() method's stack frame.
3. swapWRONG swaps firstVal and secondVal.
4. The parameters firstVal and secondVal are lost when the swapWRONG() method's stack frame is removed. The variables myVal and yourVal remain unchanged and have not been swapped.

PARTICIPATION ACTIVITY 1.15.3: Arrays and array references.

Given the code below, where are each of the following items stored?

```
int[] topScores = {100, 99, 99};
int[] bottomScores = {80, 82, 81};
int totalScores;
```

1) totalScores

- Heap
- Stack

2) topScores

- Heap
- Stack

3) topScores[0]

- Heap

4) bottomScores

- Stack
- Heap
- Stack

Passing array references

Arrays behave differently in methods because a copy of the array reference is stored in the method stack frame, and the array elements are stored on the heap. The array references in the stack frames of calling and called methods refer to the same array and have access to the array elements. Both methods can use and modify array elements. Ex: The method below determines whether two arrays of Strings contain exactly the same elements in the same order.

Figure 1.15.1: Passing array references.

```
String[] plannedCities = {"Ann Arbor", "Dexter", "Chelsea"};
String[] myVisits = {"Ann Arbor", "Chelsea", "Dexter"};

if (equals(plannedCities, myVisits)) {
   System.out.println("My visit was exactly the same as the plan.");
}

...

public static boolean equals(String[] firstArray, String[] secondArray) {
   int index;

   // If the arrays are not the same length, the arrays cannot have the
same contents
   if (firstArray.length != secondArray.length)
        return false;

   // Both arrays are now known to be the same length
   for (index = 0; index < firstArray.length; ++index) {
      if (!firstArray[index].equals(secondArray[index])) {
         return false;
      }
   }
   // Both arrays are known to be the same length and have
   // exactly the same items in the same order
   return true;
}
```

PARTICIPATION ACTIVITY 1.15.4: Methods with array parameters.

Refer to the code above.

1) The arrays plannedCities and myVisits have the same reference.
   - True
   - False

2) When the equals method is called, plannedCities and firstArray refer to the same array.
   - True
   - False

3) When the equals method is called, myVisits and firstArray have the same reference.
   - True
   - False

4) The call equals(plannedCities, myVisits) returns false.
   - True
   - False

5) The plannedCities and myVisits arrays are not altered by the equals method.
   - True
   - False

Modifying an array in a method

When an array is passed to a method, the array's reference is copied to the method's stack frame. Changes made to an array parameter in a method changes the contents of the array argument because the argument and parameter both refer to the same array. Thus, two array elements can be swapped in a method, as shown in the animation below.

PARTICIPATION ACTIVITY 1.15.5: Swapping array elements in a method.

Animation captions:

1. The heartRates array is constructed. The reference to heartRates is stored in the stack frame.
2. Passing heartRates to the swap() method copies the array reference to the parameter arrayReference. heartRates and arrayReference refer to the same array.
3. The elements at firstIndex and secondIndex are swapped.
4. The contents of heartRates have been swapped.

PARTICIPATION ACTIVITY 1.15.6: Methods modifying arrays.

Use the reverse() method to determine the value of diameterTires for each of the code segments below.

```
public static void reverse(double[] arrayReference) {
   int middle = arrayReference.length / 2; // integer division
   int index;

   // Swap mirrored elements of the array
   for (index = 0; index < middle; ++index) {
      double temp = arrayReference[index];

      // -1 below because length is unit indexed
      arrayReference[index] = arrayReference[arrayReference.length -
index - 1];
      arrayReference[arrayReference.length - index - 1] = temp;
   }
}
```

1)
```
double[] diameterTires =
{26.0, 27.3, 29.2};
reverse(diameterTires);
```

- {26.0, 27.3, 29.2}
- {29.2, 27.3, 26.0}
- The code does not compile

2)
```
double[] diameterTires =
{12.7, 42.5, 26.2, 0.0};
reverse(diameterTires);
```

- {12.7, 42.5, 26.2, 0.0}
- {0.0, 26.2, 42.5, 12.7}
- {26.2, 42.5, 12.7, 0.0}
- {26.2, 42.5, 12.7}

3)
```
double[] diameterTires = new
double[4];
diameterTires[3] = 26.5;
reverse(diameterTires);
```

- {0.0, 0.0, 0.0, 26.5}
- {0.0, 0.0, 26.5, 0.0}
- {26.5, 0.0, 0.0, 0.0}
- {0.0, 26.5, 0.0, 0.0}

4)
```
double[] diameterTires =
{13.2, 98.7, 52.8, 41.2,
18.9};
diameterTires =
reverse(diameterTires);
```

- {13.2, 98.7, 52.8, 41.2, 18.9}
- {18.9, 41.2, 52.8, 98.7, 13.2}
- The code does not compile
- The code compiles but does not run

5)
```
double[] diameterTires;
reverse(diameterTires);
```

- {0.0, 0.0, 0.0}
- A zero length array
- The code does not compile

Common error: Modifying an array reference in a method

A common error is trying to modify an array reference in a method. When an array reference is passed to a method, the reference itself is passed by value. A method cannot change the argument's array reference, since separate copies of the array reference are stored in two different stack frames. Changing the array reference in the method does not change the argument. The animation below shows an imaginative, but wrong, way to reverse an array.

PARTICIPATION ACTIVITY 1.15.7: Array references cannot be changed inside methods.

Animation captions:

1. The diameterTires array is initialized.
2. diameterTires is passed to reverseWRONG's arrayRef parameter.
3. The result array is constructed with a size equal to arrayRef's length, which is the length of diameterTires. The contents of arrayRef are copied to the result array in reverse order.
4. arrayRef is assigned with result, which modifies arrayRef in reverseWRONG's stack frame, but does not modify diameterTires in the main's stack frame.
5. When reverseWRONG's stack frame is deleted, the result array is also deleted. diameterTires has not been modified.

PARTICIPATION ACTIVITY 1.15.8: Analyze a swapReferences() method.

Consider the following code.

```
public static void swapReferences(int[] firstArray, int[] secondArray) {
   int[] tempArray = firstArray;
   firstArray = secondArray;
   secondArray = tempArray;
}

...

int[] juanitaRaces = {1, 5, 3, 2, 4};
int[] mingxiaRaces = {4, 1, 2, 3, 5};
swapReferences(juanitaRaces, mingxiaRaces);
```

1) juanitaRaces and mingxiaRaces contain the same value before the swapReferences() method is called.
   - ○ True
   - ○ False

2) Immediately after swapReferences() creates the local variable tempArray, juanitaRaces and firstArray refer to the same array.
   - ○ True
   - ○ False

3) Just before swapReferences() returns, juanitaRaces and firstArray

refer to different arrays.

- True
- False

4) The swapReference() method changed juanitaRaces.

- True
- False

5) A method cannot be used to swap two references.

- True
- False

1.16 Arrays

Array declarations and accessing elements

A programmer commonly needs to maintain a list of items, just as people often maintain lists of items like a grocery list or a course roster. An ***array*** is an ordered list of items of a given data type. Each item in an array is called an ***element***.

Construct 1.16.1: Array reference variable declaration and array allocation.

```
dataType[] arrayName = new
dataType[numElements];
```


The array declaration uses [] symbols after the data type to indicate that the variable is an array reference. An ***array reference*** variable can refer to arrays of various sizes. The ***new*** keyword creates space in memory to store the array with the specific number of elements. The array reference variable is assigned to refer to that newly allocated array. Ex:
`int[] gameScores = new int[4];` declares an array reference variable gameScores, allocates an array of four integers, and assigns gameScores to refer to the allocated array.

Terminology note: [] are ***brackets***. {} are ***braces***. In an array access, the number in brackets is

called the ***index*** of the corresponding element. The first array element is at index 0.

PARTICIPATION ACTIVITY 1.16.1: An array declaration creates multiple variables in memory, each accessible using [ ].

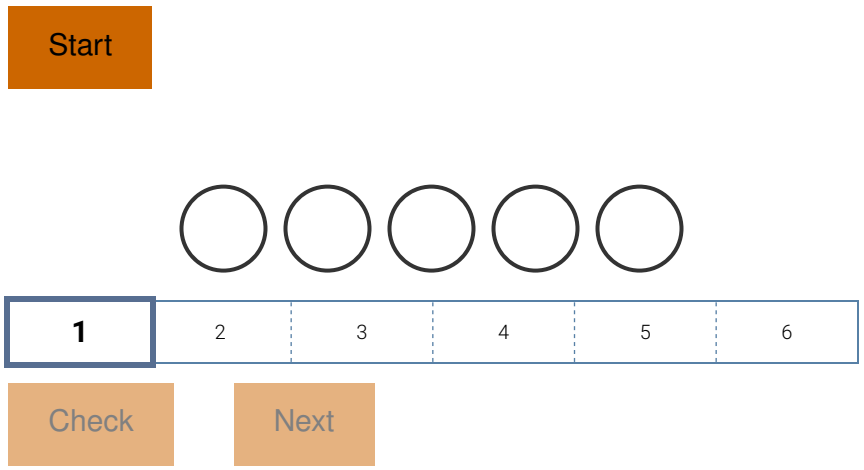
Animation content:

undefined

Animation captions:

1. An array named itemCounts is declared. The array consists of 3 elements, each of data type int. Each array element is automatically initialized with the default int value 0.
2. An element is accessed with brackets. The number in brackets is called the index of the corresponding element.

PARTICIPATION ACTIVITY 1.16.2: Select the index shown.



Separate array reference declaration and array allocation

A programmer can declare an array reference variable without allocating the array at that time and later assign the variable with an allocated array.

```
int[] gameScores;

...

gameScores = new int[4];
```


If the size of an array is known, good practice is to combine the array reference variable declaration with the array allocation.

PARTICIPATION ACTIVITY 1.16.3: Array basics.

Given:

```
int[] yearsArr = new int[4];

yearsArr[0] = 1999;
yearsArr[1] = 2012;
yearsArr[2] = 2025;
```

1) How many elements in memory does the array declaration create?

- 0
- 1
- 3
- 4

2) What value is stored in yearsArr[1]?

- 1
- 1999
- 2012

3) With what value does `currYear = yearsArr[2]` assign currYear?

- 2
- 2025
- Invalid index

4) With what value does `currYear = yearsArr[3]` assign currYear?

- 3
- 2025
- Invalid index
- 0

5) Recall that the array declaration was `int[] yearsArr = new int[4];`. Is `currYear = yearsArr[4]` a valid assignment?

- Yes, it accesses the fourth element.
- No, yearsArr[4] does not exist.

6) What is the proper way to access the *first* element in array yearsArr?

- yearsArr[1]
- yearsArr[0]

7) What are the contents of the array if the above code is followed by the statement: yearsArr[0] = yearsArr[2]?

- 1999, 2012, 1999, 0
- 2012, 2012, 2025, 0
- 2025, 2012, 2025, 0

8) What is the index of the *last* element for the following array: `int[] pricesArr = new int[100];`

- 99
- 100
- 101

Using an expression for an array index

A powerful aspect of arrays is that the index is an expression. Ex: userNums[i] uses the value held

in the int variable `i` as the index. As such, an array is useful to easily lookup the Nth item in a list.

An array's index must be an integer type. The array index cannot be a floating-point type, even if the value is 0.0, 1.0, etc.

The program below allows a user to print the age of the Nth oldest known person to have ever lived. The program quickly accesses the Nth oldest person's age using `oldestPeople[nthPerson - 1]`. Note that the index is `nthPerson - 1` rather than just `nthPerson` because an array's indices start at 0, so the 1st age is at index 0, the 2nd at index 1, etc.

Figure 1.16.1: Array's nth element can be directly accessed using [n-1]: Oldest people program.

```
import java.util.Scanner;

public class OldestPeople {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int[] oldestPeople = new int[5];
      int nthPerson;                    // User input,
Nth oldest person

      oldestPeople[0] = 122; // Died 1997 in France
      oldestPeople[1] = 119; // Died 1999 in U.S.
      oldestPeople[2] = 117; // Died 1993 in U.S.
      oldestPeople[3] = 117; // Died 1998 in Canada
      oldestPeople[4] = 116; // Died 2006 in Ecuador

      System.out.print("Enter N (1-5): ");
      nthPerson = scnr.nextInt();

      if ((nthPerson >= 1) && (nthPerson <= 5)) {
         System.out.print("The " + nthPerson + "th
oldest person lived ");
         System.out.println(oldestPeople[nthPerson -
1] + " years.");
      }
   }
}
```

```
Enter N (1-5): 1
The 1th oldest
person lived 122
years.

...

Enter N (1-5): 4
The 4th oldest
person lived 117
years.

...

Enter N (1-5): 9

...

Enter N (1-5): 0

...

Enter N (1-5): 5
The 5th oldest
person lived 116
years.
```


PARTICIPATION ACTIVITY 1.16.4: Nth oldest person program.

1) In the program above, what is the purpose of this if statement:

```
if ((nthPerson >= 1) &&
(nthPerson <= 5)) {
   ...
}
```

- To avoid overflow because nthPerson's data type can only store values from 1 to 5.
- To ensure only valid array elements are accessed because the array oldestPeople only has 5 elements.

PARTICIPATION ACTIVITY 1.16.5: Array declaration and accesses.

1) Declare and initialize an array named myVals that stores 10 elements of type int with default values.

Check Show answer

2) Assign variable x with the value stored at index 8 of array myVals.

Check Show answer

3) Assign the second element of array myVals with the value 555.

Check Show answer

4) Assign myVals array element at the index held in currIndex with the value 777.

Check Show answer

5) Assign tempVal with the myVals array element at the index one after the value held in variable i.

Check Show answer

Loops and arrays

A key advantage of arrays becomes evident when used in conjunction with loops. The program below uses a loop to allow a user to enter 8 integer values, storing those values in an array, and then printing those 8 values.

An array's ***length*** property, accessed by appending .length after the array's name, indicates the number of array elements. Ex: In the program below, `userVals.length` is 8 because the array was allocated with 8 elements.

Figure 1.16.2: Arrays combined with loops are powerful together: User-entered numbers.

```
import java.util.Scanner;

public class ArrayPrinter {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;            //
Number of elements in array
      int[] userVals = new int[NUM_ELEMENTS]; //
User numbers
      int i;                                 //
Loop index

      System.out.println("Enter " + NUM_ELEMENTS + "
integer values...");
      for (i = 0; i < userVals.length; ++i) {
         userVals[i] = scnr.nextInt();
         System.out.println("Value: " +
userVals[i]);
      }

      System.out.print("You entered: ");
      for (i = 0; i < userVals.length; ++i) {
         System.out.print(userVals[i] + " ");
      }
      System.out.println();
   }
}
```

```
Enter 8 integer
values...
Value: 5
Value: 99
Value: -1
Value: -44
Value: 8
Value: 555555
Value: 0
Value: 2
You entered: 5 99 -1
-44 8 555555 0 2
```


PARTICIPATION ACTIVITY 1.16.6: Array with loops.

Refer to the program above.

1) What is userVals.length?

- 1
- 8

2) How many times does each for loop iterate?

3) Which one line of code can be changed to allow the user to enter 100 elements?

- 1
- 8
- Unknown
- `final int NUM_ELEMENTS = 8;`
- `for (i = 0; i < userVals.length; ++i) {`

Array initialization

An array's elements are automatically initialized to default values when using the new keyword to initialize the array reference. The default value for elements of integer and floating-point data types is zero, and the default value for boolean elements is false. For information on default values of other data types, see The Java Language Specification.

A programmer may initialize an array's elements with non-default values by specifying the initial values in braces {} separated by commas. Ex: `int[] myArray = {5, 7, 11};` creates an array of three integer elements with values 5, 7, and 11. Such array initialization does not require the use of the new keyword, because the array's size is automatically set to the number of elements within the braces. For larger arrays, initialization may be done by first defining the array, and then using a loop to assign array elements.

PARTICIPATION ACTIVITY 1.16.7: Array initialization.

1) Write a single statement to declare an array of ints named myVals with 4 elements each initialized to 10.

Check Show answer

2) Given: int[] maxScores = {20, 20, 100, 50};. What is maxScores[3]?

Check Show answer

CHALLENGE ACTIVITY 1.16.1: Enter the output for the array.

422352.2723990.qx3zqy7

Start

Type the program's output


```
public class arrayOutput {
   public static void main (String [] args) {
      final int NUM_ELEMENTS = 3;
      int [] userVals = new int[NUM_ELEMENTS];
      int i;

      userVals[0] = 2;
      userVals[1] = 5;
      userVals[2] = 9;

      for (i = 0; i < userVals.length; ++i) {
         System.out.println(userVals[i]);
      }
   }
}
```

```
2
5
9
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY 1.16.2: Printing array elements.

Write three statements to print the first three elements of array runTimes. Follow each statement with a newline. Ex: If runTimes = {800, 775, 790, 805, 808}, print:

```
800
775
790
```


Note: These activities will test the code with different test values. This activity will perform two tests, both with a 5-element array. See "How to Use zyBooks".

Also note: If the submitted code tries to access an invalid array element, such as runTimes[9] for a 5-element array, the test may generate strange results. Or the test may crash and report "Program end never reached", in which case the system doesn't print the test case that caused the reported message.

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class PrintRunTimes {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        final int NUM_ELEMENTS = 5;
7        int [] runTimes = new int[NUM_ELEMENTS];
8        int i;
9
10       for (i = 0; i < runTimes.length; ++i) {
11          runTimes[i] = scnr.nextInt();
12       }
13
14       /* Your solution goes here  */
15
16    }
17 }
```


Run

View your last submission

CHALLENGE ACTIVITY 1.16.3: Printing array elements with a for loop.

Write a for loop to print all elements in courseGrades, following each element with a space (including the last). Print forwards, then backwards. End each loop with a newline. Ex: If courseGrades = {7, 9, 11, 10}, print:

```
7 9 11 10
10 11 9 7
```

Hint: Use two for loops. Second loop starts with i = courseGrades.length - 1. (Notes)

Note: These activities may test code with different test values. This activity will perform two tests, both with a 4-element array. See "How to Use zyBooks".

Also note: If the submitted code tries to access an invalid array element, such as courseGrades[9] for a 4-element array, the test may generate strange results. Or the test may crash and report "Program end never reached", in which case the system doesn't print the test case that caused the reported message.

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class CourseGradePrinter {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        final int NUM_VALS = 4;
7        int [] courseGrades = new int[NUM_VALS];
8        int i;
9
10       for (i = 0; i < courseGrades.length; ++i) {
11          courseGrades[i] = scnr.nextInt();
12       }
13
14       /* Your solution goes here  */
15
16    }
17 }
```


Run

View your last submission

1.17 Iterating through arrays

Iterating through an array using loops

Iterating through arrays using loops is commonplace and is an important programming skill to master.

Because array indices are numbered 0 to N - 1 rather than 1 to N, programmers commonly use this for loop structure:

Figure 1.17.1: Common for loop structure for iterating through an array.


```
// Iterating through myArray
for (i = 0; i < myArray.length; ++i)
{
   // Loop body accessing myArray[i]
}
```

Note that index variable i is initialized to 0, and the loop expression is i < myArray.length rather than

i <= myArray.length. If N were 5, the loop's iterations would set i to 0, 1, 2, 3, and 4, for a total of 5 iterations. The benefit of the loop structure is that each array element is accessed as myArray[i] rather than the more complex myArray[i - 1].

PARTICIPATION ACTIVITY 1.17.1: Iterating through an array.

1) Complete the code to print all items for the given array, using the above common loop structure.

```
int[] daysList = new int[365];
...
for (i = 0;
                    ; ++i) {

System.out.println(daysList[i]);
}
```

Check Show answer

2) Given that this loop iterates over all items of the array, how many items are in the array?

```
for (i = 0; i < 99; ++i) {

System.out.println(someArray[i]);
}
```

Check Show answer

Determining a quantity about an array's items

Iterating through an array for various purposes is an important programming skill to master. The program below computes the sum of an array's element values. For computing the sum, the program initializes a variable sum to 0, then simply adds the current iteration's array element value to that sum.

Figure 1.17.2: Iterating through an array example: Program that computes the sum of an array's elements.

```
import java.util.Scanner;

public class ArraySum {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;              // Number
of elements
      int[] userVals = new int[NUM_ELEMENTS]; // User
numbers
      int i;                                   // Loop
index
      int sumVal;                              // For
computing sum

      // Prompt user to populate array
      System.out.println("Enter " + NUM_ELEMENTS + "
integer values...");

      for (i = 0; i < userVals.length; ++i) {
         userVals[i] = scnr.nextInt();
         System.out.println("Value: " + userVals[i]);
      }

      // Determine sum
      sumVal = 0;
      for (i = 0; i < userVals.length; ++i) {
         sumVal = sumVal + userVals[i];
      }
      System.out.println("Sum: " + sumVal);
   }
}
```

```
Enter 8 integer
values...
Value: 3
Value: 5
Value: 234
Value: 346
Value: 234
Value: 73
Value: 26
Value: -1
Sum: 920

...

Enter 8 integer
values...
Value: 3
Value: 5
Value: 234
Value: 346
Value: 234
Value: 73
Value: 26
Value: 1
Sum: 922
```


Finding the maximum value in an array

Programs commonly iterate through arrays to determine some quantity about the array's items. The program below determines the maximum value in a user-entered list. If the user enters numbers 7, -9, 55, 44, 20, -400, 0, 2, then the program will output "max: 55". The program uses the variable maxVal to store the largest value seen "so far" as the program iterates through the array. During each iteration, if the array's current element value is larger than the max seen so far, the program assigns maxVal with the array element.

Before entering the loop, maxVal must be initialized to some value because max will be compared with each array element's value. A logical error would be to initialize maxVal to 0, because 0 is not in fact the largest value seen so far, and would result in incorrect output (of 0) if the user entered all

negative numbers. Instead, the program peeks at an array element (using the first element, though any element could have been used) and initializes maxVal to that element's value.

Figure 1.17.3: Iterating through an array example: Program that finds the max item.


```
import java.util.Scanner;

public class ArrayMax {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;              // Number
of elements
      int[] userVals = new int[NUM_ELEMENTS]; // Array
of user numbers
      int i;                                   // Loop
index
      int maxVal;                              //
Computed max

      // Prompt user to populate array
      System.out.println("Enter " + NUM_ELEMENTS + "
integer values...");

      for (i = 0; i < userVals.length; ++i) {
         userVals[i] = scnr.nextInt();
         System.out.println("Value: " + userVals[i]);
      }

      // Determine largest (max) number
      maxVal = userVals[0];                    //
Largest so far

      for (i = 0; i < userVals.length; ++i) {
         if (userVals[i] > maxVal) {
            maxVal = userVals[i];
         }
      }
      System.out.println("Max: " + maxVal);
   }
}
```

```
Enter 8 integer
values...
Value: 3
Value: 5
Value: 23
Value: -1
Value: 456
Value: 1
Value: 6
Value: 83
Max: 456

...

Enter 8 integer
values...
Value: -5
Value: -10
Value: -44
Value: -2
Value: -27
Value: -9
Value: -27
Value: -9
Max: -2
```


PARTICIPATION ACTIVITY 1.17.2: Array iteration.

Given an integer array myVals of size N_SIZE (i.e. int[] myVals = new int[N_SIZE]), complete the code to achieve the stated goal.

1) Determine the minimum number in the array, using the same initialization as the maximum number example above.

```
minVal =          ;
for (i = 0; i <
myVals.length; ++i) {
   if (myVals[i] < minVal)
{
      minVal = myVals[i];
   }
}
```

Check Show answer

2) Count how many negative numbers exist in the array.

```
cntNeg = 0;
for (i = 0; i <
myVals.length; ++i) {
   if (                )
{
      ++cntNeg;
   }
}
```

Check Show answer

3) Count how many odd numbers exist in the array.

```
cntOdd = 0;
for (i = 0; i <
myVals.length; ++i) {
   if ( (myVals[i] % 2) ==
1 ) {
                ;
   }
}
```

Check Show answer

zyDE 1.17.1: Print the sum and average of an array's elements.

Modify the program to print the average (mean) as well as the sum. Hint: You don't ac have to change the loop, but rather change what you print.

Load default template...

```
1  import java.util.Scanner;
2
3  public class ArraySum {
4     public static void main(String[] args) {
5        Scanner scnr = new Scanner(System.in);
6        final int NUM_ELEMENTS = 8;
7        int[] userVals = new int[NUM_ELEMENTS];
8        int i;
9        int sumVal;
10
11       // Prompt user to populate array
12       System.out.println("Enter " + NUM_ELE
13
14       for (i = 0; i < userVals.length; ++i)
15          userVals[i] = scnr.nextInt();
16          System.out.println("Value: " + use
17       }
```

3 5 234 346 234 73 26 -1

Run

zyDE 1.17.2: Print selected elements of an array.

Modify the program to instead just print each number that is greater than 21.

Load default template...

```
1  import java.util.Scanner;
2
3  public class ArraySum {
4     public static void main(String[] args) {
5        Scanner scnr = new Scanner(System.in);
6        final int NUM_ELEMENTS = 8;
7        int[] userVals = new int[NUM_ELEMENTS];
8        int i;
9        int sumVal;
10
11       // Prompt user to populate array
12       System.out.println("Enter " + NUM_ELE
13
14       for (i = 0; i < userVals.length; ++i)
15          userVals[i] = scnr.nextInt();
16          System.out.println("Value: " + use
17       }
```

3 5 234 346 234 73 26 -1

Run

Common error: Accessing out of range array element

A common error is to try to access an array with an index that is out of the array's index range. Ex: Trying to access highScores[8] when highScores's valid indices are 0-7. Care should be taken whenever a user enters a number that is then used as an array index, and when using a loop index as an array index also, to ensure the index is within the array's valid index range. Checking whether an array index is in range is very important. Trying to access an array with an out-of-range index results in a runtime error that causes the program to terminate.

PARTICIPATION ACTIVITY 1.17.3: Writing to an out-of-range index using an array.

Animation captions:

1. An array of 3 integers named userWeights is declared.
2. Access to indices 0, 1, and 2 are valid.
3. When array element userWeights[3] is accessed, a runtime error (i.e., exception) is thrown.

PARTICIPATION ACTIVITY 1.17.4: Iterating through an array.

Given the following code:

```
final int NUM_ELEMENTS = 5;
int[] myArray = new int[NUM_ELEMENTS];
int i;
```

1) The normal for loop structure iterates as long as: `i <= myArray.length`
   - ○ True
   - ○ False

2) To compute the sum of elements, a reasonable statement preceding the for loop is: `sumVal = 0;`
   - ○ True
   - ○ False

3) To find the maximum element value, a reasonable statement preceding the for loop is: `maxVal = 0;`

True

CHALLENGE ACTIVITY 1.17.1: Enter the output for the array.

422352.2723990.qx3zqy7

Start

Type the program's output

```
public class arrayOutput {
   public static void main (String [] args) {
      final int NUM_ELEMENTS = 3;
      int [] userVals = new int[NUM_ELEMENTS];
      int i;

      userVals[0] = 2;
      userVals[1] = 5;
      userVals[2] = 8;

      for (i = 0; i < userVals.length; ++i) {
         System.out.println(userVals[i]);
      }
   }
}
```

2
5
8

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 1.17.2: Finding values in an array.

Assign numMatches with the number of elements in userValues that equal matchValue. userValues has NUM_VALS elements. Ex: If userValues is {2, 1, 2, 2} and matchValue is 2, then numMatches should be 3.

Your code will be tested with the following values:
matchValue: 2, userValues: {2, 1, 2, 2} (as in the example program above)
matchValue: 0, userValues: {0, 0, 0, 0}
matchValue: 10, userValues: {20, 50, 70, 100}
(Notes)

422352.2723990.qx3zqy7

```
1 import java.util.Scanner;
```

Run

View your last submission

CHALLENGE ACTIVITY 1.17.3: Populating an array with a for loop.

Write a for loop to populate array userGuesses with NUM_GUESSES integers. Read integers using Scanner. Ex: If NUM_GUESSES is 3 and user enters 9 5 2, then userGuesses is {9, 5, 2}.

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class StoreGuesses {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        final int NUM_GUESSES = 3;
7        int[] userGuesses = new int[NUM_GUESSES];
8        int i;
9
10       /* Your solution goes here  */
11
12       for (i = 0; i < userGuesses.length; ++i){
13          System.out.print(userGuesses[i] + " ");
14       }
15    }
16 }
```


Run

View your last submission

CHALLENGE ACTIVITY 1.17.4: Array iteration: Sum of excess.

Array testGrades contains NUM_VALS test scores. Assign sumExtra with 0. Then, write a for loop that sets sumExtra to the total extra credit received. Full credit is 100, so anything over 100 is extra credit. Ex: If testGrades = {101, 83, 107, 90}, then sumExtra = 8, because 1 + 0 + 7 + 0 is 8.

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class SumOfExcess {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        final int NUM_VALS = 4;
7        int[] testGrades = new int[NUM_VALS];
8        int i;
9        int sumExtra; // Assign sumExtra with 0 before your for loop
10
11       for (i = 0; i < testGrades.length; ++i) {
12          testGrades[i] = scnr.nextInt();
13       }
14
15       /* Your solution goes here  */
16
17       System.out.println("sumExtra: " + sumExtra);
```

Run

View your last submission

CHALLENGE ACTIVITY 1.17.5: Printing array elements separated by commas.

Write a for loop to print all NUM_VALS elements of array hourlyTemp. Separate elements with a comma and space. Ex: If hourlyTemp = {90, 92, 94, 95}, print:


```
90, 92, 94, 95
```

Your code's output should end with the last element, without a subsequent comma, space, or newline.

422352.2723990.qx3zqy7

```
1  import java.util.Scanner;
2
3  public class PrintWithComma {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        final int NUM_VALS = 4;
7        int[] hourlyTemp = new int[NUM_VALS];
8        int i;
9
10       for (i = 0; i < hourlyTemp.length; ++i) {
11          hourlyTemp[i] = scnr.nextInt();
12       }
13
14       /* Your solution goes here  */
15
16       System.out.println("");
17    }
```


Run

View your last submission

1.18 Two-dimensional arrays

An array can be declared with two dimensions. `int[][] myArray = new int[R][C]` represents a table of int variables with R rows and C columns, so R*C elements total. For example, `int[][] myArray = new int[2][3]` creates a table with 2 rows and 3 columns, for 6 int variables total. Example accesses are `myArray[0][0] = 33;` or `num = myArray[1][2]`.

PARTICIPATION ACTIVITY 1.18.1: Two-dimensional array.

Animation content:

`undefined`

Animation captions:

1. Conceptually, a two-dimensional array is a table with rows and columns.

Conceptually, a two-dimensional array is a table with rows and columns. However, a two-dimensional array is implemented as an array of arrays. Ex:
`int[][] myArray = new int[2][3]` allocates a 2-element array, where each array element

is itself a 3 element array.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.18.1: Using a two-dimensional array: A driving distance between cities example.

```
import java.util.Scanner;

/* Direct driving distances between cities, in miles */
/* 0: Boston  1: Chicago  2: Los Angeles */
public class CityDist {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int cityA;                                  // Starting city
      int cityB;                                  // Destination city
      int [][] drivingDistances = new int[3][3]; // Driving distances

      // Initialize distances array
      drivingDistances[0][0] = 0;
      drivingDistances[0][1] = 960;  // Boston-Chicago
      drivingDistances[0][2] = 2960; // Boston-Los Angeles
      drivingDistances[1][0] = 960;  // Chicago-Boston
      drivingDistances[1][1] = 0;
      drivingDistances[1][2] = 2011; // Chicago-Los Angeles
      drivingDistances[2][0] = 2960; // Los Angeles-Boston
      drivingDistances[2][1] = 2011; // Los Angeles-Chicago
      drivingDistances[2][2] = 0;

      System.out.println("0: Boston 1: Chicago  2: Los Angeles");

      System.out.print("Enter city pair (Ex: 1 2) -- ");
      cityA = scnr.nextInt();
      cityB = scnr.nextInt();

      if ((cityA >= 0) && (cityA <= 2) && (cityB >= 0) && (cityB <= 2)) {
         System.out.print("Distance: " + drivingDistances[cityA][cityB]);
         System.out.println(" miles.");
      }
   }
}
```

```
0: Boston 1: Chicago
2: Los Angeles
Enter city pair (Ex: 1
2) -- 1 2
Distance: 2011 miles.

...

0: Boston 1: Chicago
2: Los Angeles
Enter city pair (Ex: 1
2) -- 2 0
Distance: 2960 miles.

...

0: Boston 1: Chicago
2: Los Angeles
Enter city pair (Ex: 1
2) -- 1 1
Distance: 0 miles.
```

A programmer can initialize a two-dimensional array's elements during declaration using nested braces, as below. Multiple lines make the rows and columns more visible.

Construct 1.18.1: Initializing a two-dimensional array during declaration.

```
// Initializing a 2D array
int[][] numVals = { {22, 44, 66}, {97, 98, 99}
};

// Use multiple lines to make rows more
visible
int[][] numVals = {
      {22, 44, 66}, // Row 0
      {97, 98, 99}  // Row 1
};
```


Arrays of three or more dimensions can also be declared, as in:
`int[][][] myArray = new int[2][3][5]`, which declares a total of 2*3*5 or 30 elements. Note the rapid growth in size -- an array declared as
`int[][][][] myArray = new int[100][100][5][3]` would have 100*100*5*3 or 150,000 elements. A programmer should make sure not to unnecessarily occupy available memory with a large array.

PARTICIPATION ACTIVITY 1.18.2: Two-dimensional arrays.

1) Declare and initialize a two dimensional array of integers named dataVals with 4 rows and 7 columns using default element values.

Check Show answer

2) How many total integers elements are in an array with 4 rows and 7 columns?

Check Show answer

3) How many elements are in the array declared as: char[][] streetName = new char[20][50];

Check Show answer

4) Write a statement that assigns 99 into the fifth row, third column of array numVals. Note: the first row/column is at index 0, not 1.

Check Show answer

CHALLENGE ACTIVITY 1.18.1: Find 2D array max and min.

Find the maximum value and minimum value in milesTracker. Assign the maximum value to maxMiles, and the minimum value to minMiles.

Ex: If the input is:

```
-10 20 30 40
```

the output is:

```
Min miles: -10
Max miles: 40
```

(Notes)

422352.2723990.qx3zqy7

```
1 import java.util.Scanner;
2
3 public class ArraysKeyValue {
4    public static void main (String [] args) {
5       Scanner scnr = new Scanner(System.in);
6       final int NUM ROWS = 2:
```

Run

View your last submission

1.19 File input

Opening and reading from a file

Sometimes a program should get input from a file rather than from a user typing on a keyboard. To read file input, a programmer can create a new input stream that comes from a file, rather than the predefined input stream System.in that comes from the standard input (keyboard). An input stream can then be used just like the familiar Scanner and System.in combination.

The statement `fileByteStream = new FileInputStream(str);` creates a file input stream and opens the file denoted by a String variable, str, for reading. FileInputStream's constructor also allows a programmer to pass the filename as a String literal. Ex: `fileByteStream = new FileInputStream("numFile.txt");`

PARTICIPATION ACTIVITY 1.19.1: Input from a file.

Animation content:

undefined

Animation captions:

1. The statements import java.io.FileInputStream; and import java.io.IOException; enable the use of the FileInputStream and IOException classes.
2. New stream and Scanner variables, FileInputStream fileByteStream and Scanner inFS, are

declared. The FileInputStream class, derived from InputStream, allows a programmer to read bytes from a file.

3. fileByteStream = new FileInputStream("numFile.txt"); opens the file for reading. inFS = new Scanner (fileByteStream) creates a new Scanner using fileByteStream.
4. If the open fails, either because the file does not exist or is in use by another program, main() throws an IOException and the program terminates.
5. A successfully opened input stream and Scanner object can be used to read from the file. inFS.nextInt() reads an integer into fileNum1.
6. When done using the stream, the program closes the file and input stream using fileByteStream.close().

A common error is a mismatch between the variable data type and the file data. Ex: If the data type is int but the file data is "Hello".

PARTICIPATION ACTIVITY 1.19.2: File input.

1) What is the error in the following code?

```
FileInputStream fbStream;
Scanner inFS;
int[] num;
int numElem = 0;
int i = 0;
inFS = new Scanner(fbStream);
numElem = inFS.nextInt();
num = new int[numElem];
```

- The file stream is not big enough.
- The file stream has not been properly opened.
- The nextInt() method cannot be used here.
- The code is fine.

2) Which statement opens a file inputfile.txt given `FileInputStream fbs = null;`

- `fbs.FileInputStream("inputfile.txt");`
- `fbs("inputfile.txt");`
- `fbs = new FileInputStream("inputfile.txt");`

Reading until the end of the file

A program can read varying amounts of data in a file by using a loop that reads until valid data is unavailable or the end of the file has been reached. The ***hasNextInt()*** method returns true if an integer is available for reading. If the next item in the file is not an integer or if the previous stream operation reached the end of the file, the method returns false. The Scanner class offers multiple hasNext() methods for various data types such as int, double, String, etc.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.19.1: Reading a varying amount of data from a file.

```
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.IOException;

public class FileReadVaryingAmount {
   public static void main(String[] args) throws IOException {
      FileInputStream fileByteStream = null; // File input stream
      Scanner inFS = null;                   // Scanner object
      int fileNum;                           // Data value from file

      // Try to open file
      System.out.println("Opening file myfile.txt.");
      fileByteStream = new FileInputStream("myfile.txt");
      inFS = new Scanner(fileByteStream);

      // File is open and valid if we got this far (otherwise exception
thrown)
      System.out.println("Reading and printing numbers.");

      while (inFS.hasNextInt()) {
         fileNum = inFS.nextInt();
         System.out.println("num: " + fileNum);
      }

      // Done with file, so try to close it
      System.out.println("Closing file myfile.txt.");
      fileByteStream.close(); // close() may throw IOException if fails
   }
}
```

myfile.txt with variable number of integers:

```
111
222
333
444
555
```

```
Opening file myfile.txt.
Reading and printing numbers.
num: 111
num: 222
num: 333
num: 444
num: 555
Closing file myfile.txt.
```


PARTICIPATION ACTIVITY 1.19.3: File input.

1) Which statement determines if the next line of a file has an integer available to be read?

```
fileNum =
inFS.nextInt();
if (inFS.hasNextInt())
```

Example: Counting instances of a specific word

The following program uses both the hasNext() and next() methods to determine how many times a user entered word (type String) appears in a file. The number of words in the file is unknown, so the program extracts words until no more words exist in the file.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.19.2: How many times a word appears in a file.

Program	Example input file and output
	wordFile.txt with a list of words:

```
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.IOException;

public class CountingWords {
   public static void main(String[] args) throws
IOException {
      Scanner scnr = new Scanner(System.in);
      FileInputStream fileByteStream = null; //
File input stream
      Scanner inFS = null;                   //
Scanner object
      String userWord;
      int wordFreq = 0;
      String currWord;

      // Try to open file
      System.out.println("Opening file
wordFile.txt.");
      fileByteStream = new
FileInputStream("wordFile.txt");
      inFS = new Scanner(fileByteStream);

      // Word to be found
      System.out.print("Enter a word: ");
      userWord = scnr.next();

      while (inFS.hasNext()) {
         currWord = inFS.next();
         if(currWord.equals(userWord)) {
            ++wordFreq;
         }
      }

      System.out.println(userWord + " appears in
the file " +
                         wordFreq + " times.");

      // Done with file, so try to close it
      fileByteStream.close(); // close() may
throw IOException if fails
   }
}
```

```
twenty
associable
twenty
unredacted
associable
folksay
twenty
```

```
Opening file
wordFile.txt.
Enter a word: twenty
twenty appears in the
file 3 times.
```

PARTICIPATION ACTIVITY 1.19.4: Counting instances of specific word example.

1) In the example above, how many times does the while loop iterate?
   - 3
   - 4
   - 7

2) What indicates that the end of the input file stream has been reached?
   - The hasNext() method returning false
   - The hasNext() method returning true

3) If the user entered "associable" as the userWord, how many times would the while loop execute?
   - 2
   - 7

Example: Business reviews

The following example reads a file with business reviews as the program starts and outputs data from the file at the end of the program. The number of reviews is unknown to the program, so the program continues to read until the end of the file. Each entry contains the username of the person who left the review and a 1 - 5 rating (1 being a low rating and 5 being a high rating). Upon completion, the program outputs the data from the file along with the average business rating.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 1.19.3: Program that reads business reviews from a file and computes the average rating.

Program

ReviewSystem.java

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.IOException;

public class ReviewSystem {

   public static String
readReviews(ArrayList<String> userNames,

ArrayList<Integer> userRatings)
                                    throws
IOException {
      FileInputStream fileByteStream = null; //
File input stream
      Scanner inFS = null;                   //
Scanner object
      String restaurantName;
      String userName;
      int userRating;

      // Try to open file
      System.out.println("Opening file
Trattoria_Reviews.txt.");
      fileByteStream = new
FileInputStream("Trattoria_Reviews.txt");
      inFS = new Scanner(fileByteStream);

      restaurantName = inFS.nextLine();

      while(inFS.hasNext()) {
         userName = inFS.next();
         userRating = inFS.nextInt();

         userNames.add(userName);
         userRatings.add(userRating);
      }

      fileByteStream.close(); // close() may
throw IOException if fails

      return restaurantName;
   }

   public static double
calcAvgRating(ArrayList<Integer> userRatings) {
```

Example input file and output

Trattoria_Reviews.txt with a list of usernames and ratings:

```
Trattoria Italian
Bistro

sunny8trophy
4

Angelcopter
2

Mogoodid24
5

Elixirnoel8626
5

gobbledygook
1

Friderday912
3
```

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
      int i;
      double ratingAvg = 0;

      for(i = 0; i < userRatings.size(); ++i) {
         ratingAvg += userRatings.get(i);
      }
      return ratingAvg /= userRatings.size();
   }

   public static void displayReviews(String
restaurantName,

ArrayList<String> userNames,

ArrayList<Integer> userRatings,
                                      double
ratingAvg) {
      int i;

      System.out.println("\n" + restaurantName);
      System.out.println("Average rating: " +
ratingAvg);

System.out.println("--------------------");

      for(i = 0; i < userNames.size(); ++i) {
         System.out.println("User name: " +
userNames.get(i));
         System.out.println("   Rating: " +
userRatings.get(i));
         System.out.println();
      }
   }

   public static void main(String [] args)
throws IOException {
      Scanner scnr = new Scanner(System.in);
      double ratingAvg;
      ArrayList<String> userNames = new
ArrayList<String>();
      ArrayList<Integer> userRatings = new
ArrayList<Integer>();
      String restaurantName;

      // Reads restaurant name and reviews from
input file at program start
      restaurantName = readReviews(userNames,
userRatings);

      ratingAvg = calcAvgRating(userRatings);
      displayReviews(restaurantName, userNames,
userRatings, ratingAvg);
   }
}
```



```
Trattoria Italian
Bistro
Average rating:
3.3333333333333335
--------------------
User name:
sunny8trophy
   Rating: 4

User name: Angelcopter
   Rating: 2

User name: Mogoodid24
   Rating: 5

User name:
Elixirnoel8626
   Rating: 5

User name:
gobbledygook
   Rating: 1

User name:
Friderday912
   Rating: 3
```

PARTICIPATION ACTIVITY 1.19.5: Reading and using data from a file.

Refer to the program above.

1) Which statement reads a full line of multiple strings from an input file?
   - lineString = inFS.next();
   - lineString = inFS.nextLine();

2) How is data read from the file stored in the program?
   - In two arrays
   - In two ArrayLists
   - The data is read and output immediately without being stored.

Exploring further:

- Oracle's Java FileInputStream class specification
- Oracle's Java IOException class specification

1.20 File output

Opening and writing to a file

A FileOutputStream is a class that supports writing to a file. The FileOutputStream class inherits from OutputStream.

After declaring a variable of type FileOutputStream, a file is opened using the FileOutputStream's constructor, which can take a file name string as an argument. The constructor throws an exception if the file cannot be opened for writing.

FileOutputStream only contains methods for writing bytes. To write strings and other common data types, a PrintWriter is commonly used. A PrintWriter has methods such as print() and println() and can be constructed from any OutputStream.

Table 1.20.1: Basic steps for opening and writing a file.

Action	Sample code
Open the file helloWorld.txt for writing	`FileOutputStream fileStream = new FileOutputStream("helloWorld.txt");`
Create a PrintWriter to write to the file	`PrintWriter outFS = new PrintWriter(fileStream);`
Write the string "Hello World!" to the file	`outFS.println("Hello World!");`
Close the file after writing all desired data	`outFS.close();`

PARTICIPATION ACTIVITY 1.20.1: Opening and writing to a file.

1) The FileOutputStream constructor takes _____.

- 0 arguments
- a String for a file name as an argument
- a PrintWriter as an argument

2) Given the following code, which correctly initializes outFS to enable a programmer to write to outfile.txt using PrintWriter's methods?

```
FileOutputStream outputStream =
   new FileOutputStream("outfile.txt");
PrintWriter outFS = null;
```

- `outFS = new PrintWriter(outputStream);`
- `outFS = PrintWriter(outputStream);`
- `outFS = new FileOutputStream(outputStream);`

3) The PrintWriter _____.

- is required to write to a FileOutputStream
- opens a second file for writing
- is used to allow data types other than byte arrays to be easily written to the file

Example: Writing a text file

The FileOutputStream constructor throws an exception if the output file cannot be opened. If no exception is thrown, the file is created and is initially empty. Data is written to the file, then the file is closed.

PARTICIPATION ACTIVITY 1.20.2: Writing to an output text file.

Animation content:

undefined

Animation captions:

1. A FileOutputStream named fileStream is constructed to write to myoutfile.txt. The file is initially empty.
2. An OutputStream only supports writing bytes, so a PrintWriter object is created to allow use of the println() method.
3. If no exception is thrown by FileOutputStream, then the file is open and ready for writing.
4. Two lines of text are written to the file, then the file is closed.

PARTICIPATION ACTIVITY 1.20.3: Writing a text file.

1) An exception is thrown if "myoutfile.txt" cannot be opened.

- True
- False

2) Since no data is written, the code below never creates the myoutfile.txt file on disk.

```
fileStream = new
FileOutputStream("myoutfile.txt");
fileStream.close();
```

- True
- False

3) A FileOutputStream should be closed using the close() method.

- True
- False

Example: Writing a simple HTML file

HTML files are used for web pages and consist of text content. Therefore, an HTML file can be written similar to a text file.

PARTICIPATION ACTIVITY 1.20.4: Writing simple html to a file and to the console.

Animation content:

undefined

Animation captions:

1. Output file simple.html is opened for writing.
2. A PrintWriter can be created from anything that inherits from OutputStream, such as FileOutputStream. filePrinter is created to write to the file.
3. writeHTMLFile() writes HTML content to the file using the PrintWriter. Then the output file is closed.
4. A PrintWriter can also be created from System.out, since System.out is an OutputStream. The same content is written to the console.

PARTICIPATION ACTIVITY 1.20.5: Writing a simple HTML file.

1) The PrintWriter constructor _____.

- has a parameter of type OutputStream
- has no parameters

2) A PrintWriter can be created from

either fileStream or System.out because _____.

- both are instances of FileOutputStream
- both are instances of a class that inherits from OutputStream
- both are instances of PrintWriter

3) println() is a method of _____.

- the OutputStream class
- the PrintWriter class
- both the OutputStream and PrintWriter classes

1.21 Handling exceptions

Unhandled exceptions

An ***exception*** is an unexpected incident that stops the normal execution of a program. Ex: Dividing by zero or getting invalid input results in an exception. A program that does not handle an exception ends execution.

PARTICIPATION ACTIVITY 1.21.1: An unhandled exception can end program execution.

Animation content:

undefined

Animation captions:

1. Scanner's nextDouble() method expects the user to enter a floating-point number.
2. The input "twenty" is a word instead of a floating-point number. So, scnr.nextDouble() throws an InputMismatchException exception, and program execution stops.
3. Because the exception is not handled, the program outputs the exception type and method call stack, and the program execution ends.

PARTICIPATION ACTIVITY 1.21.2: Unhandled exceptions.

Given the program below:

```
import java.util.Scanner;

public class ScanTwoInts {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int num1;
      int num2;

      System.out.print("Enter two integers: ");
      num1 = scnr.nextInt();
      num2 = scnr.nextInt();

      // ...
   }
}
```


1) Which input causes an exception?

- 12 34 two
- 43 -7
- 17 two 3

2) Which input causes an exception?

- 5
  13
- 35 45.1
- 12 34

3) Which input causes an exception?

- +5 0
- 1,0 23
- 1,000 23000

Catching exceptions

To avoid having a program end when an exception occurs, a program can use try and catch blocks to handle the exception during program execution.

- A ***try block*** surrounds normal code, which is exited immediately if a statement within the try block throws an exception.

- A ***catch block*** catches an exception thrown in a preceding try block. If the thrown exception's type matches the catch block's parameter type, the code within the catch block executes. A catch block is called an ***exception handler***.

PARTICIPATION ACTIVITY 1.21.3: Catching exceptions with try and catch blocks.

Animation content:

`undefined`

Animation captions:

1. A try block surrounds code that may throw an exception. Scanner's nextDouble() method may throw an exception if the user's input is not a floating-point number.
2. A catch block following a try block is used to catch exception during execution. The catch block's parameter type InputMismatchException indicates that the catch block only catches exceptions of that type.
3. If an InputMismatchException exception is thrown, the catch block's statements execute and print the message "Must enter a number!".
4. scnr.nextDouble() throws an InputMismatchException because the input "twenty" is a word, not a number. Execution jumps immediately to the catch block.

PARTICIPATION ACTIVITY 1.21.4: Catching exceptions.

What does the program below output for each input sequence?

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class ScanTwoInts {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int num1 = 0;
      int num2 = 0;
      int resultVal = 0;

      try {
         num1 = scnr.nextInt();
         num2 = scnr.nextInt();

         System.out.print("Good! ");
      }
      catch (InputMismatchException e) {
         System.out.print("Bad! ");
      }

      resultVal = num1 + num2;

      System.out.print(resultVal);
   }
}
```


1) 1 3

Check Show answer

2) 2 one

Check Show answer

3) ten ten

Check Show answer

Handling exceptions

A program may be able to resolve some exceptions. The previous example only printed the caught exception. Instead, a program can discard the current input line and get the distance from the user

again, as shown in the below example.

Figure 1.21.1: Handling exceptions due to invalid input.

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class LightTravelTime {
   public static void main(String[] args) {
      Scanner scnr = new
Scanner(System.in);
      double distMiles = 0.0;
      double lightTravelTime = 0.0;
      boolean needInput = true;

      while (needInput) {
         System.out.print("Enter a distance
in miles: ");

         try {
            distMiles = scnr.nextDouble();
            lightTravelTime = distMiles /
186282.0;
            needInput = false;
         }
         catch (InputMismatchException e) {
            scnr.nextLine(); // Throw away
incorrect input
         }
      }

      System.out.println("Light travels " +
distMiles +
                         " miles in " +
lightTravelTime +
                         " seconds");
   }
}
```

```
Enter a distance in miles:
1million
Enter a distance in miles: one
million
Enter a distance in miles:
1,000,000.0
Light travels 1000000.0 miles in
5.368205194275346 seconds
```


PARTICIPATION ACTIVITY 1.21.5: Handling input exceptions: restaurant max occupancy tracker.

Arrange the following lines to make a program that determines when the number of people in a res
occupants. The program continually gets the number of people entering or leaving the restaurant.
entered, and -3 means three people left. After each input, the program outputs the number of peop
number of people in the restaurant equals or exceeds 10, the program exits.

If an InputMismatchException exception occurs, the program should get and discard a single strin
abc 8" should result in 10 occupants. Not all lines are used in the solution.

Mouse: Drag/drop
Keyboard: Grab/release Spacebar (or Enter). Move ↑ ↓ ← →. Cancel Esc

Unused

```
}
System.out.println("Error");
scnr.next();
scnr.nextLine();
System.out.println("Occupancy: " + totalNumPeople);
totalNumPeople += scnr.nextInt();
try {
catch {
catch (InputMismatchException e) {
}
}
while (totalNumPeople < maxNumPeople) {
```

MaxOccupancyTracker.java

```
import java.util.Scanner;
import java.util.InputMismatchE

public class MaxOccupancyTracke
   public static void main(Stri
      Scanner scnr = new Scanne
      int maxNumPeople = 10;
      int totalNumPeople = 0;


      System.out.println("We're
   }
}
```

Check

Table 1.21.1: Common exception types.

Type	Reason exception is thrown
EOFException	End of file or end of stream has been reached unexpectedly during input
InputMismatchException	Received input does not match expected type or the input is out of range for the expected type (thrown by Scanner)
ArrayIndexOutOfBoundsException	An array has been accessed with an illegal index (negative or greater than array size)
FileNotFoundException	Attempt to open a file denoted by a filename failed
ArithmeticException	Arithmetic condition failed (Ex: Divide by zero error)

Source: Java™ Platform API Specification

CHALLENGE ACTIVITY 1.21.1: Exception handling.

422352.2723990.qx3zqy7

Start

Type the program's output

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class PowerResistorExceptHandling {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int currentLevel;
      int powerResistor;
      int resistorOhms = 1;
```

Input

2

1.22 Throwing exceptions

Using throw statements

A program can throw user-defined exceptions using a throw statement. A ***throw*** statement throws a throwable object, like an exception, during program execution. Ex: The statement `throw new Exception("Invalid date.");` creates and throws an exception with the message "Invalid date".

PARTICIPATION ACTIVITY 1.22.1: Throwing exceptions with the throw statement.

Animation content:

undefined

Animation captions:

1. Some floating-point calculations, such as 0.0 / 0.0, evaluate to "Not a Number" (NaN). A program can use Double.isNaN() to check if a floating-point variable is NaN.
2. If mixRatio is NaN, the program creates and throws a new Exception object with the message "mixRatio is NaN!".
3. Try and catch blocks surround code that may throw exceptions to handle the exceptions during program execution.
4. If the mixRatio is NaN, the throw statement executes and throws the exception. The catch block catches the exception and outputs the exception message via Exception's getMessage() method.

PARTICIPATION ACTIVITY 1.22.2: Throwing Exception objects.

1) Which statement throws an Exception

object with the message "Invalid ID"?

- `throw Exception("Invalid ID");`
- `throw new Exception();`
- `throw new Exception("Invalid ID");`

2) Which statement outputs only the message of an exception object named `excpt`?

- `System.out.println(excpt.toString());`
- `System.out.println(excpt.getMessage());`
- `System.out.println(excpt);`

3) Which catch block catches the exception thrown by the statement `throw new Exception("0.0 / 0.0 is not a number");`

- ```
  catch (Exception excpt) {
     // ...
  }
  ```
- ```
  catch
  (InputMismatchException
  excpt) {
     // ...
  }
  ```
- ```
  catch
  (ArithmeticException
  excpt) {
     // ...
  }
  ```

Using exceptions to separate error checking from normal code

A programmer can detect errors and throw exceptions to keep error-checking code separate from normal code and to reduce redundant error checks. The program below computes the density of an object by taking the ratio of mass and volume inputs. If either input is negative, the program throws an exception to handle the error.
  ```

Figure 1.22.1: Density example with error-checking code.

```
import java.util.Scanner;

public class DensityCalculator {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      double massVal = 0;    // Object mass (kg)
      double volumeVal = 0; // Object volume (m^3)
      double densityCalc;    // Resulting density

      try {
         massVal = scnr.nextDouble();

         // Error checking, non-negative mass
         if (massVal < 0.0) {
            throw new Exception("Invalid mass");
         }

         volumeVal = scnr.nextDouble();

         // Error checking, non-negative volume
         if (volumeVal < 0.0) {
            throw new Exception("Invalid volume");
         }

         densityCalc = massVal / volumeVal;

         System.out.print("Density: " + densityCalc);
      }
      catch (Exception excpt) {
         // Prints the error message passed by the throw
statement.
         System.out.print(excpt.getMessage());
      }
   }
}
```

```
3.0 2.0
Density: 1.5

...

-1.0 2.0
Invalid mass

...

3.0 -1.0
Invalid
volume
```


PARTICIPATION ACTIVITY 1.22.3: Tracing execution of the DensityCalculator program.

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

What does the above DensityCalculator program output for each input sequence?

1) 0.0 10.0

Check Show answer

2) 0.0 0.0

Check Show answer

3) -20.0 -10.0

Check Show answer

4) 0.0 -10.0

Check Show answer

Multiple exception handlers

Code within a try block may throw different types of exceptions. The example below uses multiple exception handlers to catch two different types of exceptions.

PARTICIPATION ACTIVITY 1.22.4: Multiple exception handlers.

Animation content:

undefined

Animation captions:

1. A try block can have multiple catch blocks. Each catch block handles a different type of exception.
2. The throw statement executes because circleArea is negative. Execution jumps immediately to the first catch block, which catches InputMismatchException types, not Exception types.
3. Execution jumps to the next catch block until the exception is caught or the program ends. An exception handler that catches Exception types can catch any exception because Exception is the base type.

PARTICIPATION ACTIVITY 1.22.5: Multiple exception handlers.

1) Consider the above CircleRadiusCalculator program. What is output when the input is the word pi?

- Invalid input.
- Invalid area.
- Invalid area.
  Invalid input.
- 1.0

2) Which handler arrangement catches exceptions of type Exception and InputMismatchException?

- ```
  catch
  (InputMismatchException
  excpt1, Exception
  execpt2) {
     // ...
  }
  ```

- ```
  catch
  (InputMismatchException)
  {
     // ...
  }
  catch (Exception) {
     // ...
  }
  ```

- ```
  catch (Exception excpt) {
     // ...
  }
  ```

3) Which handler arrangement can catch exceptions of type Exception and InputMismatchException in separate catch blocks?
  ```

```
catch (Exception excpt) {
   // ...
}
catch
(InputMismatchException
excpt) {
   // ...
}
```



```
catch
(InputMismatchException
excpt) {
   // ...
}
catch (Exception excpt) {
   // ...
}
```

Example: BMI calculator with multiple exception handlers

The program below computes a person's BMI from weight and height inputs. If any input is negative, the program throws an exception of type Exception. If any input is not an integer, Scanner's nextInt() method throws an exception of type InputMismatchException.

Figure 1.22.2: BMI example with multiple exception handlers.

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class BMICalculator {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int weightVal = 0;    // User defined weight
(lbs)
      int heightVal = 0;    // User defined height
(in)
      double bmiCalc;        // Resulting BMI

      try {
         System.out.print("Enter weight (in
pounds): ");
         weightVal = scnr.nextInt();

         // Error checking, non-negative weight
         if (weightVal < 0) {
            throw new Exception("Invalid weight.");
         }

         System.out.print("Enter height (in
inches): ");
         heightVal = scnr.nextInt();

         // Error checking, non-negative height
         if (heightVal < 0) {
            throw new Exception("Invalid height.");
         }

         bmiCalc = ((double) weightVal /
                    (double) (heightVal *
heightVal)) * 703.0f;

         System.out.println("BMI: " + bmiCalc);
      }
      catch (InputMismatchException excpt) {
         System.out.println("Expected a number as
input.");
         System.out.println("Cannot compute BMI.");
      }
      catch (Exception excpt) {
         // Prints the error message passed by the
throw statement.
         System.out.println(excpt.getMessage());
         System.out.println("Cannot compute BMI.");
      }
   }
}
```

```
Enter weight (in
pounds): 150
Enter height (in
inches): 66
BMI: 24.207988980716255

...

Enter weight (in
pounds): -1
Invalid weight.
Cannot compute BMI.

...

Enter weight (in
pounds): 150
Enter height (in
inches): sixty
Expected a number as
input.
Cannot compute BMI.
```

PARTICIPATION ACTIVITY 1.22.6: Handling multiple exceptions: vending machine example.

The following program simulates a vending machine panel. The program gets an integer that repre and then outputs a dispensing message. Only 8 items are for purchase. Ex: The input 2 results in t 2".

Arrange the following lines to handle two exceptions. If the user enters an item number that is not throws an Exception with the message "Try again", outputs the exception message, and then tries integer input, the program throws an InputMismatchException, outputs "Fatal error", and then exits

Not all lines are used in the solution.

Mouse: Drag/drop
Keyboard: Grab/release **Spacebar** (or **Enter**). Move ↑ ↓ ← →. Cancel **Esc**

Unused

```
askForInput = true;
askForInput = false;
catch (Exception excpt) {
}
System.out.println("Fatal error");
System.out.println(excpt);
catch (InputMismatchException excpt) {
System.out.println(excpt.getMessage());
System.out.println(excpt.toString());
}
```

VendingMachine.java

```
import java.util.Scanner;
import java.util.InputMismatchE

public class VendingMachine {
   public static void main (Str
      Scanner scnr = new Scanne
      int itemNumber = 0;
      boolean askForInput = tru

      while (askForInput) {
         try {
            itemNumber = scnr.n
            if ((itemNumber <=
               throw new Except
            }

            System.out.println(
            askForInput = false
         }
      }
```

Check

1.23 Derived classes

Derived class concept

Commonly, one class is similar to another class but with some additions or variations. Ex: A store

inventory system might use a class called GenericItem that has itemName and itemQuantity data members. But for produce (fruits and vegetables), a ProduceItem class with data members itemName, itemQuantity, and expirationDate may be desired.

PARTICIPATION ACTIVITY 1.23.1: Creating a ProduceItem from GenericItem.

Animation content:

undefined

Animation captions:

1. A GenericItem has data members itemName and itemQuantity and 3 member functions.
2. A ProduceItem is very similar to a GenericItem, but a ProduceItem also needs an expirationDate data member.
3. A ProduceItem needs the same data member functions as a GenericItem plus functions to get and set the expiration date.
4. If ProduceItem is implemented as an independent class, all the data/function members from GenericItem must be copied into ProduceItem, creating lots of duplicate code.
5. If ProduceItem is implemented as a derived class, ProduceItem need only implement what is different between a GenericItem and ProduceItem.

PARTICIPATION ACTIVITY 1.23.2: Derived class concept.

1) Creating an independent class that has the same members as an existing class creates duplicate code.
   - ○ True
   - ○ False

2) Creating a derived class is generally less work than creating an independent class.
   - ○ True
   - ○ False

Inheritance

A ***derived class*** (or ***subclass***) is a class that is derived from another class, called a ***base class*** (or

superclass). Any class may serve as a base class. The derived class is said to inherit the properties of the base class, a concept called ***inheritance***. An object declared of a derived class type has access to all the public members of the derived class as well as the public members of the base class.

A derived class is declared by placing the keyword ***extends*** after the derived class name, followed by the base class name. Ex: `class DerivedClass extends BaseClass { ... }`. The figure below defines the base class GenericItem and derived class ProduceItem that inherits from GenericItem.

Figure 1.23.1: Class ProduceItem is derived from class GenericItem.

GenericItem.java

```
public class GenericItem {
   private String itemName;
   private int itemQuantity;

   public void setName(String newName) {
      itemName = newName;
   }

   public void setQuantity(int newQty) {
      itemQuantity = newQty;
   }

   public void printItem() {
      System.out.println(itemName + " " +
itemQuantity);
   }
}
```

ProduceItem.java

```
public class ProduceItem extends GenericItem {
   private String expirationDate;

   public void setExpiration(String newDate) {
      expirationDate = newDate;
   }

   public String getExpiration() {
      return expirationDate;
   }
}
```


PARTICIPATION ACTIVITY 1.23.3: Using GenericItem and ProduceItem objects.

Animation content:

undefined

Animation captions:

1. miscItem is a GenericItem, which has 2 private fields and 3 public methods.
2. Since ProduceItem is derived from GenericItem, ProduceItem inherits GenericItem's class members and adds new members implemented in ProduceItem.
3. miscItem's name and quantity are set, and miscItem is printed to the screen.
4. perishItem's name, quantity, and expiration are set. Then perishItem is printed to the screen.

PARTICIPATION ACTIVITY 1.23.4: Derived classes.

1) A class that can serve as the basis for another class is called a _____ class.

Check Show answer

2) In the figure above, how many total class members does GenericItem contain?

Check Show answer

3) In the figure above, how many total class members are unique to ProduceItem?

Check Show answer

4) Class Dwelling has fields door1, door2, door3. A class House is derived from Dwelling and has

fields wVal, xVal, yVal, zVal. How many fields does an initialization `House h = new House();` create?

Check Show answer

Inheritance scenarios

Various inheritance variations are possible:

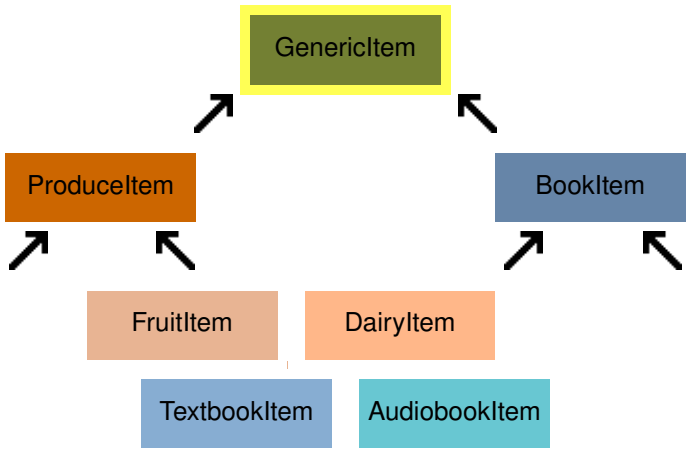
- A derived class can serve as a base class for another class. Ex: `class FruitItem extends ProduceItem {...}` creates a derived class FruitItem from ProduceItem, which was derived from GenericItem.
- A class can serve as a base class for multiple derived classes. Ex: `class FrozenFoodItem extends GenericItem {...}` creates a derived class FrozenFoodItem that inherits from GenericItem, just as ProduceItem inherits from GenericItem.

A class can only be derived from one base class directly. Ex: Inheriting from two classes as in `class House extends Dwelling, Property {...}` results in a compiler error.

PARTICIPATION ACTIVITY 1.23.5: Interactive inheritance tree.

Click a class to see available methods and data for that class.

Inheritance tree	Selected class pseudocode




```
private:
    private String itemName;
    private int itemQuantity;

public:
    public void setName(String
newName)
    public void setQuantity(int
newQty)
    public void printItem()
```

Selected class code

```
public class GenericItem {
   private String itemName;
   private int itemQuantity;

   public void setName(String newName) {
      itemName = newName;
   }

   public void setQuantity(int newQty) {
      itemQuantity = newQty;
   }

   public void printItem() {
      System.out.println(itemName + " " +
                         itemQuantity);
   }
}
```


PARTICIPATION ACTIVITY 1.23.6: Inheritance scenarios.

Refer to the interactive inheritance tree above.

1) The BookItem class acts as a derived class and a base class.
   - True
   - False

2) ProduceItem and BookItem share some of the same class members.
   - True
   - False

3) DairyItem and TextbookItem share some of the same class members.

- True
- False

4) AudiobookItem inherits the field called readerName from BookItem.

- True
- False

5) AudiobookItem inherits the method getTitle() from BookItem.

- True
- False

Example: Business and Restaurant

The example below defines a Business class with private fields name and address. The Restaurant class is derived from Business and adds a rating private field with a getter and setter.

Figure 1.23.2: Inheritance example: Business and Restaurant classes.

Business.java

```
public class Business {
   private String name;
   private String address;

   public void setName(String busName) {
      name = busName;
   }

   public void setAddress(String busAddress) {
      address = busAddress;
   }

   public String getDescription() {
      return name + " -- " + address;
   }
}
```


Restaurant.java

```
public class Restaurant extends Business {
   private int rating;

   public void setRating(int userRating) {
      rating = userRating;
   }

   public int getRating() {
      return rating;
   }
}
```

InheritanceExample.java

```
public class InheritanceExample {
   public static void main(String[] args) {
      Business someBusiness = new Business();
      Restaurant favoritePlace = new Restaurant();

      someBusiness.setName("ACME");
      someBusiness.setAddress("4 Main St");

      favoritePlace.setName("Friends Cafe");
      favoritePlace.setAddress("500 W 2nd Ave");
      favoritePlace.setRating(5);

      System.out.println(someBusiness.getDescription());
      System.out.println(favoritePlace.getDescription());
      System.out.println("  Rating: " + favoritePlace.getRating());
   }
}
```

```
ACME -- 4 Main St
Friends Cafe -- 500 W 2nd Ave
  Rating: 5
```

PARTICIPATION ACTIVITY 1.23.7: Inheritance example.

Refer to the code above.

1) How many methods are defined in Restaurant?
   - 2
   - 3
   - 5

2) How many methods can a Restaurant object call?
   - 2
   - 3
   - 5

3) Which method call produces a syntax error?
   - `someBusiness.setRating(4);`
   - `favoritePlace.getRating();`
   - `favoritePlace.setRating(4);`

4) What is the best way to declare a new DepartmentStore class?
   - `class DepartmentStore { ... }`
   - `class DepartmentStore extends Restaurant { ... }`
   - `class DepartmentStore extends Business { ... }`

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Exploring further:

- Oracle's Java tutorials on inheritance.

CHALLENGE ACTIVITY 1.23.1: Derived classes.

422352.2723990.qx3zqy7

Start

VehicleDerivation.java | Vehicle.ja

```
public class VehicleDerivation
   public static void main(Str
      Car myCar = new Car();
      myCar.setSpeed(50);

      myCar.printCarSpeed();
   }
}
```

1

Check Next

CHALLENGE ACTIVITY 1.23.2: Basic inheritance.

Assign courseStudent's name with Smith, age with 20, and ID with 9999. Use the printAll() member method and a separate println() statement to output courseStudents's data.
Sample output from the given program:

```
Name: Smith, Age: 20, ID: 9999
```

422352.2723990.qx3zqy7


```
1  // ===== Code from file Person.java =====
2  public class Person {
3     private int ageYears;
4     private String lastName;
5
6     public void setName(String userName) {
7        lastName  = userName;
8     }
9
10    public void setAge(int numYears) {
11       ageYears = numYears;
```

Run

1.24 Lab 0: Using Objects

In the file BankAccount.java, build a class called BankAccount that manages checking and savings accounts. The class has three private member fields: a customer name (String), the customer's savings account balance (double), and the customer's checking account balance (double).

Implement the following Constructor and instance methods as listed below:

- public BankAccount(String newName, double amt1, double amt2) - set the customer name to parameter newName, set the checking account balance to parameter amt1 and set the savings account balance to parameter amt2. (amt stands for amount)
- public void setName(String newName) - set the customer name
- public String getName() - return the customer name
- public void setChecking(double amt) - set the checking account balance to parameter amt
- public double getChecking() - return the checking account balance
- public void setSavings(double amt) - set the savings account balance to parameter amt
- public double getSavings() - return the savings account balance
- public void depositChecking(double amt) - add parameter amt to the checking account balance (only if positive)
- public void depositSavings(double amt) - add parameter amt to the savings account balance (only if positive)
- public void withdrawChecking(double amt) - subtract parameter amt from the checking account balance (only if positive)
- public void withdrawSavings(double amt) - subtract parameter amt from the savings account balance (only if positive)
- public void transferToSavings(double amt) - subtract parameter amt from the checking account balance and add to the savings account balance (only if positive)

422352.2723990.qx3zqy7

LAB ACTIVITY | 1.24.1: Lab 0: Using Objects | 10 / 10

File is marked as read only Current file: LabProgram.java

```
1  public class LabProgram {
2
3      public static void main(String args[]) {
4          BankAccount account = new BankAccount("Mickey", 500.00, 1000.00);
5          account.setChecking(500);
6          account.setSavings(500);
7          account.withdrawSavings(100);
8          account.withdrawChecking(100);
9          account.transferToSavings(300);
10
11         System.out.println(account.getName());
12         System.out.printf("$%.2f\n", account.getChecking());
13         System.out.printf("$%.2f\n", account.getSavings());
14     }
15 }
```


Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → **LabProgram.java** (Your program) →

Program output displayed here

Coding trail of your work What is this?

8/23 T-8 F--8,0,0,10,10 min:26

Solution Show

1.25 Program 1 - Object Oriented Programming Review

Module 1: Lab 1 - Object Oriented Programming Review

The goal of this lab is to review basic object-oriented programming concepts. We will construct and use classes, apply some inheritance to create useful hierarchies, and explore how polymorphism helps us write reusable code.

The provided Java files explore a simple use-case for object-oriented programming; a program that models people in a company. There are three classes, each of which extends from one another; we have the Manager class, which is a subclass of the Employee class, which itself is a subclass of the Person class. Essentially, an Employee is a Person, and a Manager is an Employee. Your task in this lab is to help finish these classes, as they have been left somewhat broken and incomplete.

The following .java files are included in this lab:

L1
├── Person.java
├── Employee.java
├── Manager.java
├── **Polymorphism.java ***

(This is the main class - and the **only class - that runs in zyLabs.)*

You will turn in this lab right here in zyBooks. You can test your code in Develop mode, and when you are ready to submit your code, switch to Submit mode and submit your files for grading. You are also welcome to code in your preferred environment and copy and paste your code into zyLabs to submit. Be sure to make sure your code runs correctly in zyLabs.

Here are the files for the lab if you would like to download them and work on them in another environment: L1.jar

The Person class

We start with a Person class, which simply stores a name and an age. It also has some getters, which provide access to the private name and age variables. The class should have setters too; it is your job to write those.

TO DO in this class:
- Finish the setters for name and age

For example, your toString method should return the following after you create a new Person named Jim, age 22:

```
Jim is 22 years old.
```

If you then set Jim's name to John and the age to 30, your toString should print:

```
John is 30 years old.
```


The Employee class

The Employee class extends the Person class; it is a subclass of Person. You will need to fill out another field in this class and make a new toString() method which uses the parent class's toString(). Remember that to call a parent class's methods, you can use super, as in `super.someCoolParentMethod()`.

TO DO in this class:
- Add an "employer" field to the class, and set it from the constructor
- Make a toString() method that follows the format specified in the comments

For an employee with the parameters Samir, 28, 120000, Initec, the string should be :

```
Samir is 28 years old. They make $120000 a year at Initec.
```

The Manager class

Manager is a subclass of Employee, which makes it a child of both Employee AND Person. Its constructor is broken - you will need to fix it. Remember that constructors in Java can call the parent class's constructor; you can do this by using the super() function, which directly calls the parent's constructor. Consider how it's used the Employee constructor.

TO DO in this class:
- Fix Manager's constructor so it no longer causes errors and properly initializes the object
- Change the toString() to print relevant information

For a manager with the parameters Liang, 41, 4000000000, Microsoft, the string should be:

```
Liang is a manager at Microsoft who is 41 and makes $4000000000 a
year.
```


The Polymorphism class

This class is outside the inheritance hierarchy. It's just a place to experiment with the other classes we've made so far. This is the main file that runs in zyLabs. You will notice that Person.java, Manager.java, and Employee.java have their own mains that we left in for you to test your methods

if you choose to use an IDE, but **only Polymorphism.java will run in zyLabs.** Once all of the other classes and their methods are completed, this class will run correctly.

There are a series of questions to think about in the comments in this file. Read the code and the questions carefully, and answer them to the best of your ability, giving as much justification as you can. If you get stuck, you may want to consult the all-knowing Google to top off your inheritance and polymorphism knowledge.

Submission:

To get credit for this assignment, you will need to submit the assignment under Submit mode. Once you submit, zyLabs will automatically run tests and grade your code. Below are the outputs your program should produce.

Testing Person.java :

```
Jim is 22 years old.
John is 30 years old.
```

Testing Employee:

```
Samir is 28 years old. They make $120000 a year at Initec.
```

Testing Manager:

```
Liang is a manager at Microsoft who is 41 and makes $4000000000 a year.
```

Testing Polymorphism:

This tests all of the above classes at once.

422352.2723990.qx3zqy7

LAB ACTIVITY 1.25.1: Program 1 - Object Oriented Programming Review 7 / 7

Downloadable files

`Person.java` , `Employee.java` , `Manager.java` , and `Polymorphism.java`

Download

Current file: **Person.java** Load default template...

```
1 public class Person {
```

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **Person.java** (Your program) → O

Program output displayed here

Coding trail of your work What is this?

```
8/26 F---------------2,3,3,7 min:31
```

©zyBooks 12/08/22 21:43 1361995
John Farrell
COLOSTATECS165WakefieldFall2022