2.1 Polymorphism

Polymorphism refers to determining which program behavior to execute depending on data types. Method overloading is a form of ***compile-time polymorphism*** wherein the compiler determines which of several identically-named methods to call based on the method's arguments. Another form is ***runtime polymorphism*** wherein the compiler cannot make the determination but instead the determination is made while the program is running.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

One scenario requiring runtime polymorphism involves derived classes. Programmers commonly create a collection of objects of both base and derived class types. Ex: the statement `ArrayList<GenericItem> inventoryList = new ArrayList<GenericItem>();` declares an ArrayList that can contain references to objects of type GenericItem or ProduceItem. ProduceItem derives from GenericItem.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 2.1.1: Runtime polymorphism.

The JVM can dynamically determine the correct method to call based on the object's type.

GenericItem.java:

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
public class GenericItem {
   public void setName(String newName) {
      itemName = newName;
   }

   public void setQuantity(int newQty) {
      itemQuantity = newQty;
   }

   public void printItem() {
      System.out.println(itemName + " " + itemQuantity);
   }

   protected String itemName;
   protected int itemQuantity;
}
```

ProduceItem.java:

```
public class ProduceItem extends GenericItem { // ProduceItem derived
from GenericItem
   public void setExpiration(String newDate) {
      expirationDate = newDate;
   }

   public String getExpiration() {
      return expirationDate;
   }

   @Override
   public void printItem() {
      System.out.println(itemName + " " + itemQuantity
                         + " (Expires: " + expirationDate +
")");
   }

   private String expirationDate;
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

ItemInventory.java:

```
import java.util.ArrayList;

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

```
        GenericItem genericItem1;
        ProduceItem produceItem1;
        ArrayList<GenericItem> inventoryList = new
ArrayList<GenericItem>(); // Collection of "Items"
        int i;
// Loop index

        genericItem1 = new GenericItem();
        genericItem1.setName("Smith Cereal");
        genericItem1.setQuantity(9);

        produceItem1 = new ProduceItem();
        produceItem1.setName("Apple");
        produceItem1.setQuantity(40);
        produceItem1.setExpiration("May 5, 2012");

        genericItem1.printItem();
        produceItem1.printItem();

        // More common: Collection (e.g., ArrayList) of objs
        // Polymorphism -- Correct  printItem()  called
        inventoryList.add(genericItem1);
        inventoryList.add(produceItem1);
        System.out.println("\nInventory: ");
        for (i = 0; i < inventoryList.size(); ++i) {
           inventoryList.get(i).printItem(); // Calls correct printItem()
        }
    }
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
Smith Cereal 9
Apple 40 (Expires: May 5, 2012)

Inventory:
Smith Cereal 9
Apple 40 (Expires: May 5, 2012)
```

The program uses a Java feature relating to ***derived/base class reference conversion*** wherein a reference to a derived class can be converted to a reference to the base class (without explicit casting). Such conversion is in contrast to other data type conversions, such as converting a double to an int (which is an error unless explicitly cast). Thus, the above statement `inventoryList.add(produceItem1);` uses this feature, with a ProduceItem reference being converted to a GenericItem reference (inventoryList is an ArrayList of GenericItem references). The conversion is intuitive; recall in an earlier animation that a derived class like ProduceItem consists of the base class GenericItem plus additional members, so the conversion yields a reference to the base class part (so really there's no change).

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

However, an interesting question arises when printing the ArrayList's contents. For a given element, how does the program know whether to call GenericItem's printItem() or ProduceItem's printItem()? The Java virtual machine automatically performs runtime polymorphism, i.e., it dynamically determines the correct method to call based on the actual object type to which the variable (or element) refers.

PARTICIPATION ACTIVITY 2.1.1: Polymorphism.

Consider the GenericItem and ProduceItem classes defined above.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

1) An item of type `ProduceItem` may be added to an ArrayList of type `ArrayList<GenericItem>`.
   - True
   - False

2) The JVM automatically performs runtime polymorphism to determine the correct method to call.
   - True
   - False

Exploring further:

- More on Polymorphism from Oracle's Java tutorials

CHALLENGE ACTIVITY 2.1.1: Polymorphism and method overloading.

422352.2723990.qx3zqy7

Start

Type the program's output

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

CallWatch.java | Watch.java | SmartWatch.java

```
3:1
14:
10:
```

```
import java.util.ArrayList;

public class CallWatch {
   public static void main(String[] args) {
      SmartWatch watch1;
      SmartWatch watch2;
      Watch watch3;

      ArrayList<Watch> watchList = new ArrayList<Watch>();
      int i;

      watch1 = new SmartWatch();
      watch1.setHours(10);
      watch1.setMins(42);
      watch1.setPercentage(30);

      watch2 = new SmartWatch();
      watch2.setHours(14);
      watch2.setMins(39);
      watch2.setPercentage(45);

      watch3 = new Watch();
      watch3.setHours(3);
      watch3.setMins(11);

      watchList.add(watch3);
      watchList.add(watch2);
      watchList.add(watch1);

      for(i = 0; i < watchList.size(); ++i) {
         watchList.get(i).printItem();
      }
   }
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

CHALLENGE ACTIVITY 2.1.2: Basic polymorphism.

Write the printItem() method for the base class. Sample output for below program:

```
Last name: Smith
First and last name: Bill Jones
```

422352.2723990.qx3zqy7

```
// ===== Code from file BaseItem.java =====
public class BaseItem {
   protected String lastName;

   public void setLastName(String providedName) {
      lastName = providedName;
   }

   // FIXME: Define printItem() method

   /* Your solution goes here  */
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Run

View your last submission

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

2.2 zyBooks built-in programming window

This section has been set as optional by your instructor.

zyDE 2.2.1: Programming window.

Load default template...

```
public class YourProgram {
   public static void main(String[] args) {
      // Enter your program here
   }
}
```

Pre-enter any input for program, th
run.

Run

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

2.3 The Object class

The Object class

The built-in ***Object class*** serves as the base class for all other classes and does not have a base class. All classes, including user-defined classes, are derived from Object and implement Object's methods. In the following discussion, note the subtle distinction between the term "Object class" and the generic term "object", which can refer to the instance of any class. Two common methods defined within the Object class are toString() and equals().

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

- The ***toString()*** method returns a String representation of the Object. By default, toString() returns a String containing the object's class name followed by the object's hash code in hexadecimal form. Ex: java.lang.Object@372f7a8d.

- The ***equals(otherObject)*** method compares an Object to otherObject and returns true if both variables reference the same object. Otherwise, equals() returns false. By default, equals() tests the equality of the two Object references, not the equality of the Objects' contents.

Figure 2.3.1: Business class is derived from Object.

```
public class Business {
   protected String name;
   protected String address;

   void setName(String busName) {
      name = busName;
   }

   void setAddress(String
busAddress) {
      address = busAddress;
   }

   String getDescription() {
      return name + " -- " +
address;
   }
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.3.1: Behavior of toString() method.

Animation content:

undefined

Animation captions:

1. The Object class is the base class for all other classes.
2. Java's Integer class overrides the Object's toString() method, but the user-defined Business class does not.
3. Object's toString() returns a string consisting of the object's class name (java.lang.Object), the '@' character, and an unsigned hexadecimal representation of the object's hash code (1148ab5c).
4. Integer's toString() method returns the object's associated integer value.
5. Business does not override toString() so Object's toString() method is called and outputs the object's class name (Business), the '@' character, and an unsigned hexadecimal representation of the object's hash code (19469ea2).

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.3.2: The Object class and overriding the toString() method.

1) User-defined classes are not derived from the Object class.

- True
- False

2) All classes can access Object's public and protected methods like toString() and equals(), even if the methods are not explicitly overridden.

- True
- False

3) The built-in Integer class overrides the toString() method in order to return a String representing an Integer's value.

- True
- False

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

4) The Object class's toString() method returns a String containing only the Object instance's type.

- True
- False

Overriding toString() in the base class

The figure below shows a Business class that overrides Object's toString() method and returns a String containing the business name and address. The Restaurant class derives from Business but does not override toString(). So when a Restaurant object's toString() method is called, the Business class's toString() method executes.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 2.3.2: Base class Business overrides toString().

Business.java

```
public class Business {
   protected String name;
   protected String address;

   void setName(String busName) {
      name = busName;
   }

   void setAddress(String busAddress) {
      address = busAddress;
   }

   @Override
   public String toString() {
      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;
   }
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

The toString() method is called automatically by the compiler when an object is concatenated to a string or when print() or println() is called. Ex: `System.out.println(someObj)` calls `someObj.toString()` automatically.

PARTICIPATION ACTIVITY

2.3.3: toString() in base class only.

Refer to the code below to determine what each code snippet outputs.

```
Business aaaBus = new Business();
Restaurant tacoRest = new Restaurant();

aaaBus.setName("AAA Business");
aaaBus.setAddress("5 Race St");

tacoRest.setName("Tom's Tacos");
tacoRest.setAddress("600 Pleasure Ave");
tacoRest.setRating(5);
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

1) `System.out.println(aaaBus.toString());`

- Business@372f7a8d
- AAA Business -- 5 Race St
- Tom's Tacos -- 600 Pleasure Ave

2) `System.out.println(tacoRest);`

- Restaurant@372f7a8d
- AAA Business -- 5 Race St
- Tom's Tacos -- 600 Pleasure Ave

3)
```
String somePlace = aaaBus + " #2";
System.out.println(somePlace);
```

- Syntax error
- AAA Business -- 5 Race St
- AAA Business -- 5 Race St #2

4)
```
String somePlace = aaaBus;
System.out.println(somePlace);
```

- Syntax error
- AAA Business
- AAA Business -- 5 Race St

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Overriding toString() in the derived class

Both the base class Business and derived class Restaurant override toString() in the figure below.

The Restaurant toString() uses the super keyword to call the base class toString() to get a string with the business name and address. Then toString() concatenates the rating and returns a string containing the name, address, and rating.

Figure 2.3.3: Derived class Restaurant overrides toString().

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Business.java

```
public class Business {
   protected String name;
   protected String address;

   void setName(String busName) {
      name = busName;
   }

   void setAddress(String busAddress) {
      address = busAddress;
   }

   @Override
   public String toString() {
      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;
   }

   @Override
   public String toString() {
      return super.toString() + ", Rating: " +
rating;
   }
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY | 2.3.4: toString() in base and derived classes().

Refer to the code below to determine what each code snippet outputs.

```
Business aaaBus = new Business();
Restaurant tacoRest = new Restaurant();

aaaBus.setName("AAA Business");
aaaBus.setAddress("5 Race St");

tacoRest.setName("Tom's Tacos");
tacoRest.setAddress("600 Pleasure Ave");
tacoRest.setRating(5);
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

1) `System.out.println(aaaBus.toString());`

- Business@372f7a8d
- AAA Business -- 5 Race St
- Tom's Tacos -- 600 Pleasure Ave

2) `System.out.println(tacoRest);`

- Restaurant@372f7a8d
- Tom's Tacos -- 600 Pleasure Ave
- Tom's Tacos -- 600 Pleasure Ave, Rating: 5

3)
```
String somePlace = aaaBus + " &\n" + tacoRest;
System.out.println(somePlace);
```

- Syntax error
- AAA Business -- 5 Race St & Tom's Tacos -- 600 Pleasure Ave, Rating: 5
- AAA Business & Tom's Tacos

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Exploring further:

- Oracle's Java Object class specification.
- Oracle's Java class hierarchy.

CHALLENGE ACTIVITY | 2.3.1: Enter the output of the println() statements.

422352.2723990.qx3zqy7

Start

Type the program's output

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

CallCar.java | Car.java

```
public class CallCar {
   public static void main(String[] args) {
      Car car1 = new Car();
      Car car2 = new Car();

      car1.setBrand("Nissan");
      car1.setModel("Rogue");
      car2.setBrand("Honda");
      car2.setModel("CR-V");

      System.out.println(car1);
      System.out.println(car2);
   }
}
```

```
Nissan, Rogu
Honda, CR-V
```

1 2

Check Next

2.4 Access by members of derived classes

Member access

The members of a derived class have access to the public members of the base class, but not to the private members of the base class. This is logical—allowing access to all private members of a class merely by creating a derived class would circumvent the idea of private members. Thus, adding the following member method to the Restaurant class yields a compiler error.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 2.4.1: Member methods of a derived class cannot access private members of the base class.

```
public class Business {
   private String name;
   private String address;

   ...

}

public class Restaurant extends Business {
   private int rating;

   ...

   public void displayRestaurant() {
      System.out.println(name);
      System.out.println(address);
      System.out.println("Rating: "+ rating);
   }
   ...
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
$ javac Restaurant.java
Restaurant.java:12: name has private access in Business
      System.out.println(name);
                         ^
Restaurant.java:12: address has private access in Business
      System.out.println(address);
                         ^
2 errors
```

PARTICIPATION ACTIVITY 2.4.1: Access by derived class members.

Assume `public class Restaurant extends Business{...}`

1) Business's public member method can be called by a member method of Restaurant.

- True
- False

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

2) Restaurant's private fields can be accessed by Business.

True

Protected member access

Recall that members of a class may have their access specified as *public* or *private*. A third access specifier is ***protected***, which provides access to derived classes and all classes in the same ***package*** but not by anyone else. Packages are discussed in detail elsewhere, but for our purposes a package can just be thought of as the directory in which program files are located. Thus, classes in the same package are located in the same directory. The following illustrates the implications of the protected access specifier.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

In the following example, the member called name is specified as protected and is accessible anywhere in the derived class. Note that the name member is also accessible in main()—the protected specifier also allows access to classes in the same package; protected members are private to everyone else.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 2.4.2: Access specifiers—Protected allows access by derived classes and classes in the same package but not by others.

Code contains intended errors to demonstrate protected accesses.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Business.java:

```
public class Business{
   protected String name;    // Member accessible by self and derived
classes
   private String address;   // Member accessible only by self

   public void printMembers() { // Member accessible by anyone
      // Print information ...
   }

}
```

Restaurant.java:

```
public class Restaurant extends Business{
   private int rating;

   public void displayRestaurant() {
      // Attempted accesses
      printMembers();             // OK
      name = "Gyro Hero";         // OK    ("protected" above made this
possible)
      address = "5 Fifth St";    // ERROR
   }

   // Other class members ...
}
```

InheritanceAccessEx.java

```
public class InheritanceAccessEx {
   public static void main(String[] args) {
      Business business = new Business();
      Restaurant restaurant = new Restaurant();

      // Attempted accesses
      business.printMembers();            // OK
      business.name = "Gyro Hero";        // OK (protected also applies to
other classes in the same package)
      business.address = "5 Fifth St";   // ERROR

      restaurant.printMembers();          // OK
      restaurant.name = "Gyro Hero";     // OK (protected also applies to
other classes in the same package)
      restaurant.rating = 5; // ERROR

      // Other instructions ...
   }
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

To make Restaurant's displayRestaurant() method work, we merely need to change the private members to protected members in class Business. Business's class members name and address thus become accessible to a derived class like Restaurant. A programmer may often want to make some members protected in a base class to allow access by derived classes, while making other members private to the base class.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

The following table summarizes access specifiers.

Table 2.4.1: Access specifiers for class members.

Specifier	Description
private	Accessible by self.
protected	Accessible by self, derived classes, and other classes in the same package.
public	Accessible by self, derived classes, and everyone else.
no specifier	Accessible by self and other classes in the same package.

PARTICIPATION ACTIVITY | 2.4.2: Protected access specifier.

Assume `public class Restaurant extends Business{...}`.

Suppose a new class, `public class SkateShop{...}`, is defined in the same package as Business.

1) Business's protected fields can be accessed by a member method of Restaurant.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

- True
- False

2) Business's protected fields can be accessed by a member method of SkateShop.

- True
- False

Class definitions

Separately, the keyword "public" in a class definition like `public class DerivedClass {...}` specifies a class's visibility in other classes in the program:

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

- *public* : A class can be used by every class in the program regardless of the package in which either is defined.
- *no specifier* : A class can be used only in other classes within the same package, known as ***package-private***.

Most beginning programmers define classes as public when learning to program.

PARTICIPATION ACTIVITY 2.4.3: Access specifiers for class definitions.

Suppose a new class, `RestaurantReview`, is defined in a separate package. For the following cases, which specifier, if any, should be used for
__________ `class Restaurant{...}`?

1) Restaurant can be accessed by RestaurantReview.
   - no specifier
   - public
   - protected

2) Restaurant cannot be accessed by RestaurantReview.
   - no specifier
   - public
   - protected

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Exploring further:

- More on access specifiers from Oracle's Java tutorials

2.5 Overriding member methods

Overriding

When a derived class defines a member method that has the same name and parameters as a base class's method, the member method is said to ***override*** the base class's method. The example below shows how the Restaurant's getDescription() method overrides the Business's getDescription() method.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

The ***@Override*** annotation is placed above a method that overrides a base class method so the compiler verifies that an identical base class method exists. An ***annotation*** is an optional command beginning with the "@" symbol that can provide the compiler with information that helps the compiler detect errors better. The @Override annotation causes the compiler to produce an error when a programmer mistakenly specifies parameters that are different from the parameters of the method that should be overridden or misnames the overriding method. Good practice is to always include an @Override annotation with a method that is meant to override a base class method.

PARTICIPATION ACTIVITY 2.5.1: Overriding member method example.

Animation content:

undefined

Animation captions:

1. The Business class defines a getDescription() method that returns a string with the business name and address.
2. Restaurant derives from Business and uses the @Override annotation above a member method that has the same name, parameters, and return type as the base class method getDescription().
3. The Restaurant object favoritePlace calls the Restaurant's getDescription(), which overrides the base class's getDescription().

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Overriding vs. overloading

Overriding differs from overloading. In overloading, methods with the same name must have different parameter types. In overriding, a derived class member method takes precedence over a base class member method with the same name and parameter types. Overloading is performed if derived and base member methods have different parameter types; the member method of the derived class does not hide the member method of the base class.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.5.2: Overriding.

Refer to the code above.

1) If a Restaurant object calls getDescription(), the Restaurant's getDescription() is called instead of Business's getDescription().

- True
- False

2) If a Business object calls getDescription(), the Restaurant's getDescription() is called instead of Business's getDescription().

- True
- False

3) Removing Business's getDescription() method in the example above causes a syntax error.

- True
- False

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

4) Changing Restaurant's `String getDescription()` to `String getDescription(int num)` causes a syntax error.

True

False

5) Changing Business's name and address data members from protected to private in the example above causes a syntax error.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

True

False

Calling a base class method

An overriding method can call the overridden method by using the super keyword. Ex: `super.getDescription()`. The ***super*** keyword is a reference variable used to call the parent class's methods or constructors.

Figure 2.5.1: Method calling overridden method of base class.

```
public class Restaurant extends Business{
   ...

   @Override
   public String getDescription() {
      return super.getDescription() + "\n  Rating: " +
rating;
   }

   ...
}
```

A common error is to leave off super when wanting to call a base class method. Without the use of the super keyword, the call to getDescription() refers to itself (a *recursive* call), so getDescription() would call itself, which would call itself, etc., never actually printing anything.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.5.3: Override example.

Choose the correct replacement for the missing code below so ProduceItem's printItem() overrides GenericItem's printItem().

GenericItem.java

```
public class GenericItem {
   __(A)__: String itemName;
   __(A)__: int itemQuantity;
   ...

   public void printItem() {
      System.out.println(itemName + " " + itemQuantity);
   }
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

ProduceItem.java

```
public class ProduceItem extends GenericItem {
   private String expirationDate;

   public void setExpiration(String newDate) {
      expirationDate = newDate;
   }

   public String getExpiration() {
      return expirationDate;
   }

   @Override
   public __(B)__ {
      __(C)__;
      System.out.println("  (expires " + expirationDate + ")");
   }
}
```

1) (A)

- private
- public

2) (B)

- void printItem(int itemNumber)
- void printItem()
- String printItem()

3) (C)

- printItem()
- printItem(super)
- super.printItem()

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

CHALLENGE ACTIVITY 2.5.1: Overriding member methods.

422352.2723990.qx3zqy7

Start

Type the program's output

ClassOverridingEx.java | Computer.java | Laptop.java

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
public class ClassOverridingEx {
   public static void main(String[] args) {
      Laptop myLaptop = new Laptop();

      myLaptop.setComputerStatus("30%", "connected");
      myLaptop.setWiFiStatus("bad");

      myLaptop.printStatus();

   }
}
```

```
WiFi
CPU:
```

1 2

Check Next

CHALLENGE ACTIVITY 2.5.2: Basic derived class member override.

Define a method printAll() for class PetData that prints output as follows with inputs "Fluffy", 5, and 4444. Hint: Make use of the base class' printAll() method.

```
Name: Fluffy, Age: 5, ID: 4444
```

422352.2723990.qx3zqy7

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
// ===== Code from file AnimalData.java =====
public class AnimalData {
   private int ageYears;
   private String fullName;

   public void setName(String givenName) {
      fullName = givenName;
   }

   public void setAge(int numYears) {
      ageYears = numYears;
   }

```

Run

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

2.6 Is-a versus has-a relationships

The concept of inheritance is commonly confused with the idea of composition. Composition is the idea that one object may be made up of other objects, such as a MotherInfo class being made up of objects like firstName (which may be a String object), childrenData (which may be an ArrayList of ChildInfo objects), etc. Defining that MotherInfo class does *not* involve inheritance, but rather just composing the sub-objects in the class.

Figure 2.6.1: Composition.

The 'has-a' relationship. A MotherInfo object 'has a' String object and 'has a' ArrayList of ChildInfo objects, but no inheritance is involved.

```
public class ChildInfo {
   public String firstName;
   public String birthDate;
   public String schoolName;

   ...
}

public class MotherInfo {
   public String firstName;
   public String birthDate;
   public String spouseName;
   public ArrayList<ChildInfo> childrenData;

   ...
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

In contrast, a programmer may note that a mother is a kind of person, and all persons have a name and birthdate. So the programmer may decide to better organize the program by defining a PersonInfo class, and then by creating the MotherInfo class derived from PersonInfo, and likewise for the ChildInfo class.

Figure 2.6.2: Inheritance.

The 'is-a' relationship. A MotherInfo object 'is a' kind of PersonInfo. The MotherInfo class thus inherits from the PersonInfo class. Likewise for the ChildInfo class.

```
public class PersonInfo {
   public String firstName;
   public String birthdate;

   ...
}

public class ChildInfo extends PersonInfo {
   public String schoolName;

   ...
}

public class MotherInfo extends PersonInfo {
   public String spousename;
   public ArrayList<ChildInfo> childrenData;
  ...
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.6.1: Is-a vs. has-a relationships.

Indicate whether the relationship of the everyday items is an is-a or has-a relationship. Derived classes and inheritance are related to is-a relationships, not has-a relationships.

1) Pear / Fruit

- Is-a
- Has-a

2) House / Door

- Is-a
- Has-a

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

3) Dog / Owner

- Is-an
- Has-an

4) Mug / Cup

- Is-a
- Has-a

UML diagrams

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Programmers commonly draw class inheritance relationships using ***Unified Modeling Language (UML)*** notation (IBM: UML basics).

PARTICIPATION ACTIVITY | 2.6.2: UML derived class example: ProduceItem derived from GenericItem.

Animation content:

```
undefined
```

Animation captions:

1. A class diagram depicts a class' name, data members, and methods.
2. A solid line with a closed, unfilled arrowhead indicates a class is derived from another class.
3. The derived class only shows additional members.

2.7 Selection sort

Selection sort is a sorting algorithm that treats the input as two parts, a sorted part and an unsorted part, and repeatedly selects the proper next value to move from the unsorted part to the end of the sorted part.

PARTICIPATION ACTIVITY | 2.7.1: Selection sort.

Animation content:

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
undefined
```

Animation captions:

1. Selection sort treats the input as two parts, a sorted and unsorted part. Variables i and j keep track of the two parts.

2. The selection sort algorithm searches the unsorted part of the array for the smallest element; indexSmallest stores the index of the smallest element found.
3. Elements at i and indexSmallest are swapped.
4. Indices for the sorted and unsorted parts are updated.
5. The unsorted part is searched again, swapping the smallest element with the element at i.
6. The process repeats until all elements are sorted.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

The index variable i denotes the dividing point. Elements to the left of i are sorted, and elements including and to the right of i are unsorted. All elements in the unsorted part are searched to find the index of the element with the smallest value. The variable indexSmallest stores the index of the smallest element in the unsorted part. Once the element with the smallest value is found, that element is swapped with the element at location i. Then, the index i is advanced one place to the right, and the process repeats.

The term "selection" comes from the fact that for each iteration of the outer loop, a value is selected for position i.

PARTICIPATION ACTIVITY | 2.7.2: Selection sort algorithm execution.

Assume selection sort's goal is to sort in ascending order.

1) Given list {9 8 7 6 5}, what value will be in the 0^{th} element after the first pass over the outer loop (i = 0)?

Check Show answer

2) Given list {9 8 7 6 5}, how many swaps will occur during the first pass of the outer loop (i = 0)?

Check Show answer

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

3) Given list {5 9 8 7 6} and i = 1, what will be the list after completing the second outer loop iteration? Use curly

brackets in your answer

Check Show answer

Selection sort has the advantage of being easy to code, involving one loop nested within another loop, as shown below.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 2.7.1: Selection sort algorithm.

```
public class SelectionSort {
   public static void selectionSort(int [] numbers) {
      int i;
      int j;
      int indexSmallest;
      int temp;      // Temporary variable for swap

      for (i = 0; i < numbers.length - 1; ++i) {

         // Find index of smallest remaining element
         indexSmallest = i;
         for (j = i + 1; j < numbers.length; ++j) {

            if (numbers[j] < numbers[indexSmallest]) {
               indexSmallest = j;
            }
         }

         // Swap numbers[i] and numbers[indexSmallest]
         temp = numbers[i];
         numbers[i] = numbers[indexSmallest];
         numbers[indexSmallest] = temp;
      }
   }

   public static void main(String [] args) {
      int numbers [] = {10, 2, 78, 4, 45, 32, 7, 11};
      int i;

      System.out.print("UNSORTED: ");
      for (i = 0; i < numbers.length; ++i) {
         System.out.print(numbers[i] + " ");
      }
      System.out.println();

      /* initial call to selection sort with index */
      selectionSort(numbers);

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

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
UNSORTED: 10 2 78 4 45 32 7 11
SORTED: 2 4 7 10 11 32 45 78
```

Selection sort may require a large number of comparisons. The selection sort algorithm runtime is

O(N^2). If a list has N elements, the outer loop executes N - 1 times. For each of those N - 1 outer loop executions, the inner loop executes an average of $\frac{N}{2}$ times. So the total number of comparisons is proportional to $(N-1) \cdot \frac{N}{2}$, or O(N^2). Other sorting algorithms involve more complex algorithms but have faster execution times.

PARTICIPATION ACTIVITY | 2.7.3: Selection sort runtime.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

1) When sorting a list with 50 elements, `indexSmallest` will be assigned to a minimum of _____ times.

Check Show answer

2) How many times longer will sorting a list of 20 elements take compared to sorting a list of 10 elements?

Check Show answer

3) How many times longer will sorting a list of 500 elements take compared to a list of 50 elements?

Check Show answer

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

CHALLENGE ACTIVITY | 2.7.1: Selection sort.

422352.2723990.qx3zqy7

Start

When using selection sort to sort a list with 23 elements, what is the minimum number of assignments to **`indexSmallest`**?

1	2	3	4

Check Next

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

2.8 Data structures

Data structures

A ***data structure*** is a way of organizing, storing, and performing operations on data. Operations performed on a data structure include accessing or updating stored data, searching for specific data, inserting new data, and removing data. The following provides a list of basic data structures.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Table 2.8.1: Basic data structures.

Data structure	Description
Record	A ***record*** is the data structure that stores subitems, often called fields, with a name associated with each subitem.
Array	An ***array*** is a data structure that stores an ordered list of items, where each item is directly accessible by a positional index.
Linked list	A ***linked list*** is a data structure that stores an ordered list of items in nodes, where each node stores data and has a pointer to the next node.
Binary tree	A ***binary tree*** is a data structure in which each node stores data and has up to two children, known as a left child and a right child.
Hash table	A ***hash table*** is a data structure that stores unordered items by mapping (or hashing) each item to a location in an array.
Heap	A ***max-heap*** is a tree that maintains the simple property that a node's key is greater than or equal to the node's childrens' keys. A ***min-heap*** is a tree that maintains the simple property that a node's key is less than or equal to the node's childrens' keys.
Graph	A ***graph*** is a data structure for representing connections among items, and consists of vertices connected by edges. A ***vertex*** represents an item in a graph. An ***edge*** represents a connection between two vertices in a graph.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.8.1: Basic data structures.

1) A linked list stores items in an unspecified order.

- True
- False

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

2) A node in binary tree can have zero, one, or two children.

True

False

3) A list node's data can store a record with multiple subitems.

True

False

4) Items stored in an array can be accessed using a positional index.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

True

False

Choosing data structures

The selection of data structures used in a program depends on both the type of data being stored and the operations the program may need to perform on that data. Choosing the best data structure often requires determining which data structure provides a good balance given expected uses. Ex: If a program requires fast insertion of new data, a linked list may be a better choice than an array.

PARTICIPATION ACTIVITY 2.8.2: A linked list avoids the shifting problem.

Animation content:

`undefined`

Animation captions:

1. Inserting an item at a specific location in an array requires making room for the item by shifting higher-indexed items.
2. Once the higher index items have been shifted, the new item can be inserted at the desired index.
3. To insert new item in a linked list, a list node for the new item is first created.
4. Item B's next pointer is assigned to point to item C. Item A's next pointer is updated to point to item B. No shifting of other items was required.

John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.8.3: Basic data structures.

1) Inserting an item at the end of a 999-item array requires how

many items to be shifted?

Check Show answer

2) Inserting an item at the end of a 999-item linked list requires how many items to be shifted?

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Check Show answer

3) Inserting an item at the beginning of a 999-item array requires how many items to be shifted?

Check Show answer

4) Inserting an item at the beginning of a 999-item linked list requires how many items to be shifted?

Check Show answer

2.9 Introduction to algorithms

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Algorithms

An ***algorithm*** describes a sequence of steps to solve a computational problem or perform a calculation. An algorithm can be described in English, pseudocode, a programming language, hardware, etc. A ***computational problem*** specifies an input, a question about the input that can be answered using a computer, and the desired output.

PARTICIPATION ACTIVITY | 2.9.1: Computational problems and algorithms.

Animation captions:

1. A computational problem is a problem that can be solved using a computer. A computational problem specifies the problem input, a question to be answered, and the desired output.
2. For the problem of finding the maximum value in an array, the input is an array of numbers.
3. The problem's question is: What is the maximum value in the input array? The problem's output is a single value that is the maximum value in the array.
4. The FindMax algorithm defines a sequence of steps that determines the maximum value in the array.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY | 2.9.2: Algorithms and computational problems.

Consider the problem of determining the number of times (or frequency) a specific word appears in a list of words.

1) Which can be used as the problem input?

- String for user-specified word
- Array of unique words and string for user-specified word
- Array of all words and string for user-specified word

2) What is the problem output?

- Integer value for the frequency of most frequent word
- String value for the most frequent word in input array
- Integer value for the frequency of specified word

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

3) An algorithm to solve this computation problem must be written using a programming language.

True

Practical applications of algorithms

Computational problems can be found in numerous domains, including e-commerce, internet technologies, biology, manufacturing, transportation, etc. Algorithms have been developed for numerous computational problems within these domains.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

A computational problem can be solved in many ways, but finding the best algorithm to solve a problem can be challenging. However, many computational problems have common subproblems, for which efficient algorithms have been developed. The examples below describe a computational problem within a specific domain and list a common algorithm (each discussed elsewhere) that can be used to solve the problem.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Table 2.9.1: Example computational problems and common algorithms.

Application domain	Computational problem	Common algorithm
DNA analysis	Given two DNA sequences from different individuals, what is the longest shared sequence of nucleotides?	*Longest common substring problem:* A longest common substring algorithm determines the longest common substring that exists in two input strings. DNA sequences can be represented using strings consisting of the letters A, C, G, and T to represent the four different nucleotides.
Search engines	Given a product ID and a sorted array of all in-stock products, is the product in stock and what is the product's price?	*Binary search:* The binary search algorithm is an efficient algorithm for searching a list. The list's elements must be sorted and directly accessible (such as an array).
Navigation	Given a user's current location and desired location, what is the fastest route to walk to the destination?	*Dijkstra's shortest path:* Dijkstra's shortest path algorithm determines the shortest path from a start vertex to each vertex in a graph. The possible routes between two locations can be represented using a graph, where vertices represent specific locations and connecting edges specify the time required to walk between those two locations.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.9.3: Computational problems and common algorithms.

Match the common algorithm to another computational problem that can be solved using that algorithm.

If unable to drag and drop, refresh the page.

Longest common substring **Shortest path algorithm** **Binary search**

Do two student essays share a common phrase consisting of a sequence of more than 100 letters?

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Given the airports at which an airline operates and distances between those airports, what is the shortest total flight distance between two airports?

Given a sorted list of a company's employee records and an employee's first and last name, what is a specific employee's phone number?

Reset

Efficient algorithms and hard problems

Computer scientists and programmers typically focus on using and designing efficient algorithms to solve problems. Algorithm efficiency is most commonly measured by the algorithm runtime, and an efficient algorithm is one whose runtime increases no more than polynomially with respect to the input size. However, some problems exist for which an efficient algorithm is unknown.

NP-complete problems are a set of problems for which no known efficient algorithm exists. NP-complete problems have the following characteristics:

- No efficient algorithm has been found to solve an NP-complete problem.
- No one has proven that an efficient algorithm to solve an NP-complete problem is impossible.
- If an efficient algorithm exists for one NP-complete problem, then all NP-complete problems can be solved efficiently.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

By knowing a problem is NP-complete, instead of trying to find an efficient algorithm to solve the problem, a programmer can focus on finding an algorithm to efficiently find a good, but non-optimal, solution.

PARTICIPATION ACTIVITY | 2.9.4: Example NP-complete problem: Cliques.

Animation captions:

1. A programmer may be asked to write an algorithm to solve the problem of determining if a set of K people who all know each other exists within a graph of a social network?
2. For the example social network graph and K = 3, the algorithm should return yes. Xiao, Sean, and Tanya all know each other. Sean, Tanya, and Eve also all know each other.
3. For K = 4, no set of 4 individual who all know each other exists, and the algorithm, should return no.
4. This problem is equivalent to the clique decision problem, which is NP-complete, and no known polynomial time algorithm exists.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY | 2.9.5: Efficient algorithm and hard problems.

1) An algorithm with a polynomial runtime is considered efficient.

- True
- False

2) An efficient algorithm exists for all computational problems.

- True
- False

3) An efficient algorithm to solve an NP-complete problem may exist.

- True
- False

2.10 Relation between data structures and algorithms

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Algorithms for data structures

Data structures not only define how data is organized and stored, but also the operations performed on the data structure. While common operations include inserting, removing, and searching for data, the algorithms to implement those operations are typically specific to each data structure. Ex: Appending an item to a linked list requires a different algorithm than appending an item to an array.

PARTICIPATION ACTIVITY | 2.10.1: A list avoids the shifting problem.

Animation content:

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

undefined

Animation captions:

1. The algorithm to append an item to an array determines the current size, increases the array size by 1, and assigns the new item as the last array element.
2. The algorithm to append an item to a linked list points the tail node's next pointer and the list's tail pointer to the new node.

PARTICIPATION ACTIVITY | 2.10.2: Algorithms for data structures.

Consider the array and linked list in the animation above. Can the following algorithms be implemented with the same code for both an array and linked list?

1) Append an item
- Yes
- No

2) Return the first item
- Yes
- No

3) Return the current size
- Yes
- No

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Algorithms using data structures

Some algorithms utilize data structures to store and organize data during the algorithm execution. Ex: An algorithm that determines a list of the top five salespersons, may use an array to store salespersons sorted by their total sales.

Figure 2.10.1: Algorithm to determine the top five salespersons using an array.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

```
DisplayTopFiveSalespersons(allSalespersons) {
   // topSales array has 5 elements
   // Array elements have subitems for name and total sales
   // Array will be sorted from highest total sales to lowest total sales
   topSales = Create array with 5 elements

   // Initialize all array elements with a negative sales total
   for (i = 0; i < topSales⇢length; ++i) {
      topSales[i]⇢name = ""
      topSales[i]⇢salesTotal = -1
   }

   for each salesPerson in allSalespersons {
      // If salesPerson's total sales is greater than the last
      // topSales element, salesPerson is one of the top five so far
      if (salesPerson⇢salesTotal > topSales[topSales⇢length -
1]⇢salesTotal) {

         // Assign the last element in topSales with the current
salesperson
         topSales[topSales⇢length - 1]⇢name =  salesPerson⇢name
         topSales[topSales⇢length - 1]⇢salesTotal =
salesPerson⇢salesTotal

         // Sort topSales in descending order
         SortDescending(topSales)
      }
   }

   // Display the top five salespersons
   for (i = 0; i < topSales⇢length; ++i) {
      Display topSales[i]
   }
}
```

PARTICIPATION ACTIVITY 2.10.3: Top five salespersons.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

1) Which of the following is not equal to the number of items in the topSales array?

- topSales⇢length
- 5
- allSalesperson⇢length

2) To adapt the algorithm to display the top 10 salespersons, what modifications are required?

- Only the array creation
- All loops in the algorithm
- Both the creation and all loops

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

3) If allSalespersons only contains three elements, the DisplayTopFiveSalespersons algorithm will display elements with no name and negative sales.

- True
- False

2.11 Abstract data types

Abstract data types (ADTs)

An ***abstract data type*** (***ADT***) is a data type described by predefined user operations, such as "insert data at rear," without indicating how each operation is implemented. An ADT can be implemented using different underlying data structures. However, a programmer need not have knowledge of the underlying implementation to use an ADT.

Ex: A list is a common ADT for holding ordered data, having operations like append a data item, remove a data item, search whether a data item exists, and print the list. A list ADT is commonly implemented using arrays or linked list data structures.

PARTICIPATION ACTIVITY | 2.11.1: List ADT using array and linked lists data structures.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Animation captions:

1. A new list named agesList is created. Items can be appended to the list. The items are ordered.
2. Printing the list prints the items in order.
3. A list ADT is commonly implemented using array and linked list data structures. But, a

programmer need not have knowledge of which data structure is used to use the list ADT.

PARTICIPATION ACTIVITY 2.11.2: Abstract data types.

1) Starting with an empty list, what is the list contents after the following operations?

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Append(list, 11)
Append(list, 4)
Append(list, 7)

- 4, 7, 11
- 7, 4, 11
- 11, 4, 7

2) A remove operation for a list ADT will remove the specified item. Given a list with contents: 2, 20, 30, what is the list contents after the following operation?

Remove(list, item 2)

- 2, 30
- 2, 20, 30
- 20, 30

3) A programmer must know the underlying implementation of the list ADT in order to use a list.

- True
- False

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

4) A list ADT's underlying data structure has no impact on the program's execution.

- True
- False

Common ADTs

Table 2.11.1: Common ADTs.

Abstract data type	Description	Common underlying data structures
List	A ***list*** is an ADT for holding ordered data.	Array, linked list
Dynamic array	A ***dynamic array*** is an ADT for holding ordered data and allowing indexed access.	Array
Stack	A ***stack*** is an ADT in which items are only inserted on or removed from the top of a stack.	Linked list
Queue	A ***queue*** is an ADT in which items are inserted at the end of the queue and removed from the front of the queue.	Linked list
Deque	A ***deque*** (pronounced "deck" and short for double-ended queue) is an ADT in which items can be inserted and removed at both the front and back.	Linked list
Bag	A ***bag*** is an ADT for storing items in which the order does not matter and duplicate items are allowed.	Array, linked list
Set	A ***set*** is an ADT for a collection of distinct items.	Binary search tree, hash table
Priority queue	A ***priority queue*** is a queue where each item has a priority, and items with higher priority are closer to the front of the queue than items with lower priority.	Heap
Dictionary (Map)	A ***dictionary*** is an ADT that associates (or maps) keys with values.	Hash table, binary search tree

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.11.3: Common ADTs.

Consider the ADTs listed in the table above. Match the ADT with the description of the order and uniqueness of items in the ADT.

If unable to drag and drop, refresh the page.

Set **Bag** **List** **Priority queue**

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Items are ordered based on how items are added. Duplicate items are allowed.

Items are not ordered. Duplicate items are not allowed.

Items are ordered based on items' priority. Duplicate items are allowed.

Items are not ordered. Duplicate items are allowed.

Reset

2.12 Applications of ADTs

Abstraction and optimization

Abstraction means to have a user interact with an item at a high-level, with lower-level internal details hidden from the user. ADTs support abstraction by hiding the underlying implementation details and providing a well-defined set of operations for using the ADT.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Using abstract data types enables programmers or algorithm designers to focus on higher-level operations and algorithms, thus improving programmer efficiency. However, knowledge of the underlying implementation is needed to analyze or improve the runtime efficiency.

PARTICIPATION ACTIVITY | 2.12.1: Programming using ADTs.

Animation content:

`undefined`

Animation captions:

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

1. Abstraction simplifies programming. ADTs allow programmers to focus on choosing which ADTs best match a program's needs.
2. Both the List and Queue ADTs support efficient interfaces for removing items from one end (removing oldest entry) and adding items to the other end (adding new entries).
3. The list ADT supports iterating through list contents in reverse order, but the queue ADT does not.
4. To use the List (or Queue) ADT, the programmer does not need to know the List's underlying implementation.

PARTICIPATION ACTIVITY | 2.12.2: Programming with ADTs.

Consider the example in the animation above.

1) The _____ ADT is the better match for the program's requirements.

- queue
- list

2) The list ADT _____.

- can only be implemented using an array
- can only be implemented using a linked list
- can be implemented in numerous ways

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

3) Knowledge of an ADT's underlying implementation is needed to analyze the runtime efficiency.

- True
- False

ADTs in standard libraries

Most programming languages provide standard libraries that implement common abstract data types. Some languages allow programmers to choose the underlying data structure used for the ADTs. Other programming languages may use a specific data structure to implement each ADT, or may automatically choose the underlying data-structure.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Table 2.12.1: Standard libraries in various programming languages.

Programming language	Library	Common supported ADTs
Python	Python standard library	list, set, dict, deque
C++	Standard template library (STL)	vector, list, deque, queue, stack, set, map
Java	Java collections framework (JCF)	Collection, Set, List, Map, Queue, Deque

PARTICIPATION ACTIVITY 2.12.3: ADTs in standard libraries.

1) Python, C++, and Java all provide built-in support for a deque ADT.
   - True
   - False

2) The underlying data structure for a list data structure is the same for all programming languages.
   - True
   - False

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

3) ADTs are only supported in standard libraries.
   - True
   - False

2.13 Algorithm efficiency

Algorithm efficiency

An algorithm describes the method to solve a computational problem. Programmers and computer scientists should use or write efficient algorithms. ***Algorithm efficiency*** is typically measured by the algorithm's computational complexity. ***Computational complexity*** is the amount of resources used by the algorithm. The most common resources considered are the runtime and memory usage.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY | 2.13.1: Computational complexity.

Animation captions:

1. An algorithm's computational complexity includes runtime and memory usage.
2. Measuring runtime and memory usage allows different algorithms to be compared.
3. Complexity analysis is used to identify and avoid using algorithms with long runtimes or high memory usage.

PARTICIPATION ACTIVITY | 2.13.2: Algorithm efficiency and computational complexity.

1) Computational complexity analysis allows the efficiency of algorithms to be compared.

- True
- False

2) Two different algorithms that produce the same result have the same computational complexity.

- True
- False

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

3) Runtime and memory usage are the only two resources making up computational complexity.

Runtime complexity, best case, and worst case

An algorithm's ***runtime complexity*** is a function, T(N), that represents the number of constant time operations performed by the algorithm on an input of size N. Runtime complexity is discussed in more detail elsewhere.

©zyBooks 12/08/22 21:46 1361995
COLOSTATECS165WakefieldFall2022

Because an algorithm's runtime may vary significantly based on the input data, a common approach is to identify best and worst case scenarios. An algorithm's ***best case*** is the scenario where the algorithm does the minimum possible number of operations. An algorithm's ***worst case*** is the scenario where the algorithm does the maximum possible number of operations.

Input data size must remain a variable

A best case or worst case scenario describes contents of the algorithm's input data only. The input data size must remain a variable, N. Otherwise, the overwhelming majority of algorithms would have a best case of N=0, since no input data would be processed. In both theory and practice, saying "the best case is when the algorithm doesn't process any data" is not useful. Complexity analysis always treats the input data size as a variable.

PARTICIPATION ACTIVITY | 2.13.3: Linear search best and worst cases.

Animation captions:

1. LinearSearch searches through array elements until finding the key. Searching for 26 requires iterating through the first 3 elements.
2. The search for 26 is neither the best nor the worst case.
3. Searching for 54 only requires one comparison and is the best case: The key is found at the start of the array. No other search could perform fewer operations.
4. Searching for 82 compares against all array items and is the worst case: The number is not found in the array. No other search could perform more operations.

©zyBooks 12/08/22 21:46 1361995
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY | 2.13.4: FindFirstLessThan algorithm best and worst case.

Consider the following function that returns the first value in a list that is less than the specified value. If no list items are less than the specified value, the specified value is returned.

```
FindFirstLessThan(list, listSize, value) {
   for (i = 0; i < listSize; i++) {
      if (list[i] < value)
         return list[i]
   }
   return value // no lesser value found
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

If unable to drag and drop, refresh the page.

Best case **Worst case** **Neither best nor worst case**

No items in the list are less than `value`.

The first half of the list has elements greater than `value` and the second half has elements less than `value`.

The first item in the list is less than `value`.

Reset

PARTICIPATION ACTIVITY 2.13.5: Best and worst case concepts.

1) The linear search algorithm's best case scenario is when N = 0.

- True
- False

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

2) An algorithm's best and worst case scenarios are always different.

- True
- False

Space complexity

An algorithm's ***space complexity*** is a function, S(N), that represents the number of fixed-size memory units used by the algorithm for an input of size N. Ex: The space complexity of an algorithm that duplicates a list of numbers is S(N) = 2N + k, where k is a constant representing memory used for things like the loop counter and list pointers.

Space complexity includes the input data and additional memory allocated by the algorithm. An algorithm's ***auxiliary space complexity*** is the space complexity not including the input data. Ex: An algorithm to find the maximum number in a list will have a space complexity of S(N) = N + k, but an auxiliary space complexity of S(N) = k, where k is a constant.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.13.6: FindMax space complexity and auxiliary space complexity.

Animation content:

undefined

Animation captions:

1. FindMax's arguments represent input data. Non-input data includes variables allocated in the function body: maximum and i.
2. The list's size is a variable, N. Three integers are also used, listSize, maximum, and i, making the space complexity S(N) = N + 3.
3. The auxiliary space complexity includes only the non-input data, which does not increase for larger input lists.
4. The function's auxiliary space complexity is S(N) = 2.

PARTICIPATION ACTIVITY 2.13.7: Space complexity of GetEvens function.

Consider the following function, which builds and returns a list of even numbers from the input list.

```
GetEvens(list, listSize) {
   i = 0
   evensList = Create new, empty list
   while (i < listSize) {
      if (list[i] % 2 == 0)
         Add list[i] to evensList
      i = i + 1
   }
   return evensList
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

1) What is the maximum possible size of the returned list?

- listSize
- listSize / 2

2) What is the minimum possible size of the returned list?

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

- listSize / 2
- 1
- 0

3) What is the worst case auxiliary space complexity of GetEvens if N is the list's size and k is a constant?

- S(N) = N + k
- S(N) = k

4) What is the best case auxiliary space complexity of GetEvens if N is the list's size and k is a constant?

- S(N) = N + k
- S(N) = k

2.14 ArrayLists of Objects

Because all classes are derived from the Object class, programmers can take advantage of runtime polymorphism in order to create a collection (e.g., ArrayList) of objects of various class types and perform operations on the elements. The program below uses the Business class and other built-in classes to create and output a single ArrayList of differing types.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 2.14.1: Business.java.

```
public class Business {
   protected String name;
   protected String address;

   public Business() {}

   public Business(String busName, String
busAddress) {
      name = busName;
      address = busAddress;
   }

   @Override
   public String toString() {
      return name + " -- " + address;
   }
}
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

PARTICIPATION ACTIVITY 2.14.1: ArrayList of Objects.

Animation content:

undefined

Animation captions:

1. objList is an ArrayList of Object elements. All objects derive from Object, so objList can store any type of object.
2. Five new objects of various class types are added to the ArrayList. Each derived class reference is automatically converted to a base class (Object) reference.
3. printArrayList() takes an ArrayList of Objects as an argument and outputs every element of the ArrayList.
4. get(i) returns each Object element. Runtime polymorphism allows the correct version of toString() to be called based on the actual class type of each element.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Note that a method operating on a collection of Object elements may only invoke the methods defined by the base class (e.g., the Object class). Thus, a statement that calls the toString() method on an element of an ArrayList of Objects called objList, such as `objList.get(i).toString()`, is valid because the Object class defines the toString() method. However, a statement that calls, for example, the Integer class's intValue() method on the same element (i.e., `objList.get(i).intValue()`) results in a compiler error even if that particular element is an Integer object.

PARTICIPATION ACTIVITY | 2.14.2: ArrayLists of Object elements and runtime polymorphism principles.

1) An item of *any* class type may be added to an ArrayList of type `ArrayList<Object>`.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

- Yes
- No

2) Assume that an ArrayList of type `ArrayList<Object>` called myList contains only three elements of type Double. Is the statement `myList.get(0).doubleValue();` valid?

Note that the method doubleValue() is defined in the Double class but not the Object class.

- Yes
- No

3) The above program's printArrayList() method can dynamically determine which implementation of toString() to call.

- Yes
- No

CHALLENGE ACTIVITY | 2.14.1: Enter the output of the ArrayList of Objects.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

422352.2723990.qx3zqy7

Start

Type the program's output

ArrayListManager.java | Business.java | Coffee.java

```
import java.util.ArrayList;

public class ArrayListManager {

   public static void printArrayList(ArrayList<Object> objList) {
      int i;

      for (i = 0; i < objList.size(); ++i) {
         System.out.println(objList.get(i));
      }
   }

   public static void main(String[] args) {
      ArrayList<Object> objList = new ArrayList<Object>();

      String myString = new String("Echo");
      Coffee myCoffee = new Coffee("Latte", "French");
      Business myBusiness = new Business("ACME", "5 Main St");

      objList.add(myCoffee);
      objList.add(myString);
      objList.add(myBusiness);

      printArrayList(objList);
   }
}
```

```
Latte (
Echo
ACME --
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

1

Check Try again

Exploring further:

- Oracle's Java Object class specification.
- More on Polymorphism from Oracle's Java tutorials

2.15 Java example: Employee list using ArrayLists

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

This section has been set as optional by your instructor.

zyDE 2.15.1: Managing an employee list using an ArrayList.

The following program allows a user to add to and list entries from an ArrayList, whic maintains a list of employees.

1. Run the program, and provide input to add three employees' names and related Then use the list option to display the list.
2. Modify the program to implement the deleteEmployee method.
3. Run the program again and add, list, delete, and list again various entries.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Load default ter

```
import java.util.ArrayList;
import java.util.Scanner;

public class EmployeeManager {
   static Scanner scnr = new Scanner(System.in);

   public static void main(String[] args) {
      final int MAX_ELEMENTS = 10;
      final char EXIT_CODE = 'X';
      final String PROMPT_ACTION = "Add, Delete, List or eXit (a,d,l,x): ";
      ArrayList<String> name       = new ArrayList<String>(MAX_ELEMENTS);
      ArrayList<String> department = new ArrayList<String>(MAX_ELEMENTS);
      ArrayList<String> title      = new ArrayList<String>(MAX_ELEMENTS);
      char userAction;

      // Loop until the user enters the exit code.
      userAction = getAction(PROMPT_ACTION);
```

```
a
Rajeev Gupta
Sales
```

Run

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Below is a solution to the above problem.

zyDE 2.15.2: Managing an employee list using an ArrayList (solution).

Load default ter

```
import java.util.ArrayList;
import java.util.Scanner;

public class MemoryManagement {
   static Scanner scnr = new Scanner(System.in);

   public static void main(String[] args) {
      final int MAX_ELEMENTS = 10;
      final char EXIT_CODE = 'X';
      final String PROMPT_ACTION = "Add, Delete, List, or eXit (a,d,l,x): ";
      ArrayList<String> name       = new ArrayList<String>(MAX_ELEMENTS);
      ArrayList<String> department = new ArrayList<String>(MAX_ELEMENTS);
      ArrayList<String> title      = new ArrayList<String>(MAX_ELEMENTS);
      char userAction;

      // Loop until the user enters the exit code
      userAction = getAction(PROMPT_ACTION);
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

a
Rajeev Gupta
Sales

Run

2.16 Lab 1: Remove all even numbers from an array

Write the removeEvens() method, which receives an array of integers as a parameter and returns a new array of integers containing only the odd numbers from the original array. The main program outputs values of the returned array.

John Farrell
COLOSTATECS165WakefieldFall2022

Hint: If the original array has even numbers, then the new array will be smaller in length than the original array and should have no blank elements.

Ex: If the array passed to the removeEvens() method is [1, 2, 3, 4, 5, 6, 7, 8, 9], then the method returns and the program output is:

```
[1, 3, 5, 7, 9]
```

Ex: If the array passed to the removeEvens() method is [1, 9, 3], then the method returns and the program output is:

```
[1, 9, 3]
```

422352.2723990.qx3zqy7

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

LAB ACTIVITY | 2.16.1: Lab 1: Remove all even numbers from an array | 10 / 10

LabProgram.java

Load default template...

```java
import java.util.Arrays;

public class LabProgram {

   public static int[] removeEvens(int [] nums) {

      /* Type your code here */
      int newSize = 0;


            for ( int x : nums ) {
                  if ( x%2 != 0 ) newSize++;
            }

            int[] newArray = new int[newSize];

            int count = 0;
```

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.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Run program

Input (from above) → **LabProgram.java** (Your program) →

Program output displayed here

Coding trail of your work What is this?

```
8/25 R----------------10 min:19
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

2.17 Lab 2: Pet information (derived classes)

The base class `Pet` has private fields petName, and petAge. The derived class `Dog` extends the `Pet` class and includes a private field for dogBreed. Complete main() to:

- create a generic pet and print information using printInfo().
- create a `Dog` pet, use printInfo() to print information, and add a statement to print the dog's breed using the getBreed() method.

Ex. If the input is:

```
Dobby
2
Kreacher
3
German Schnauzer
```

the output is:

```
Pet Information:
   Name: Dobby
   Age: 2
Pet Information:
   Name: Kreacher
   Age: 3
   Breed: German Schnauzer
```

422352.2723990.qx3zqy7

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

LAB ACTIVITY | 2.17.1: Lab 2: Pet information (derived classes) 10 / 10

File is marked as read only Current file: **Dog.java**

```
public class Dog extends Pet {
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

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) → **Dog.java** (Your program) → 0

Program output displayed here

Coding trail of your work What is this?

8/30 T----------------------0,0----0,10 min:42

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

2.18 LAB: Descending selection sort with output during execution

Write a void method selectionSortDescendTrace() that takes an integer array, and sorts the array into descending order. The method should use nested loops and output the array after each iteration of the outer loop, thus outputting the array N-1 times (where N is the size). Complete main() to read in a list of up to 10 positive integers (ending in -1) and then call the selectionSortDescendTrace() method.

If the input is:

```
20 10 30 40 -1
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

then the output is:

```
40 10 30 20
40 30 10 20
40 30 20 10
```

422352.2723990.qx3zqy7

LAB ACTIVITY | 2.18.1: LAB: Descending selection sort with output during execution 10 / 10

DescendingOrder.java Load default template...

```
import java.util.Scanner;

public class DescendingOrder {
   // TODO: Write a void method selectionSortDescendTrace() that takes
   //       an integer array and the number of elements in the array as arguments,
   //       and sorts the array into descending order.
   public static void selectionSortDescendTrace(int [] numbers, int numElements) {
        int tempNumber;
        int x = 0;
        int numberOfElements = numElements;
        int largestNumIndex = 0;
        int[] workingArray = numbers;

        for ( x=0; x<numberOfElements-1; x++ ) {

            largestNumIndex = x;

```

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.

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Enter program input (optional)

20 10 30 40

Run program

Input (from above) → DescendingOrder.java (Your program)

Program output displayed here

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Coding trail of your work What is this?

```
9/3 S-----------0,0-0---0,10 min:27
```

©zyBooks 12/08/22 21:46 1361995
John Farrell
COLOSTATECS165WakefieldFall2022