Inheritance is a fundamental technique used to create and organize reusable classes. The child is-a more specific version of parent. The child inherits characteristics of the parent (methods and data defined by the parent class). Tailor derived class by adding new variables or methods, or by modifying the inherited ones. The keyword extends is used to establish an is-a (inheritance) relationship. The “canonical” example, introduced by Alan Kay. You’ll see it in Lab!

Assignment #2 is due Monday at 11:59 PM!
No points off for javadoc, but do try it!

Do not miss SI sessions!
The next lab is super cool!

Reading for today and next time: Ch. 9 and 10 (with some exceptions)
- A child class can be the parent of another child, forming a **class hierarchy**
- Two children of the same parent are called **siblings**
- Common features should be put as high in the hierarchy as is reasonable
- An inherited member is passed continually down the line
  - Therefore, a child class inherits from all its ancestor classes
protected

- A `protected` variable is visible to any class in the same package as the parent class.
- The `protected` modifier allows a child class to reference a variable or method directly in the parent class.
- It provides more **encapsulation** than `public` visibility, but is not as tightly encapsulated as `private` visibility.

super

- Constructors are **not** inherited, even though they have public visibility.
  - Yet, we often want to use the parent's constructor to set up the "parent's part" of the object.
- The keyword `super` can be used to refer to the parent class, including the parent's constructor.
- A child's constructor should:
  - Call the parent's constructor as its first line: `super();`
  - If it does not call `super()`, a 0-parameters `super()` constructor will be called anyway!
- The `super` reference can also be used to reference other variables & methods defined in parent's class.

```java
/**
 * Demonstrates the use of an inherited method.
 * @Author Java Foundations
 */
public class Words2 {

/**
 * Driver: Instantiates a derived class and invokes its inherited
 * and local methods.
 */
public static void main (String[] args) {
    Dictionary<String> webster = new Dictionary<String>(2500, 52500);
    System.out.println("Number of pages: " + webster.getNumPages());
    System.out.println("Number of definitions: " + webster.getNumDefinitions());
    System.out.println("Definitions per page: " + webster.computeDefRatio());
}
}
```

```java
/**
 * Represents a book. Used as the parent of a derived class to
 * demonstrate inheritance and the use of the super reference.
 * @Author Java Foundations
 */
public class Book2 {
    protected int pages;

    // Constructor (will be invoked by child)
    public Book2 (int numPages) {
        pages = numPages;
    }

    /**
     * @Param number of pages in the book
     */
    public void setPages (int numPages) {
        pages = numPages;
    }

    /**
     * @Return number of pages in the book
     */
    public int getPages () {
        return pages;
    }
}
```
A HungerGamesDictionary object keeps track of definitions related to terms in the Hunger Games (such as the word "mockingjay") as well as the number of districts in Panem.

Write a HungerGamesDictionary class that inherits from Dictionary2 and contains the following:

- An integer instance variable that keeps track of the number of districts
- A constructor with three parameters corresponding to the number of pages, the number of definitions, and the number of districts
- A toString method that returns a tab-delimited String consisting of the number of pages, the number of definitions, and the number of districts

```java
public class HungerGamesDictionary extends Dictionary2 {
    private int definitions;
    
    public HungerGamesDictionary(int numPages, int numDefinitions, int numDistricts) {
        super(numPages);
        definitions = numDefinitions;
    }
    
    public double computeRatio() {
        return definitions/pages;
    }
    
    public void setDefinitions(int numDefinitions) {
        definitions = numDefinitions;
    }
    
    public int getDefinitions() {
        return definitions;
    }
}
```

A Thought class demonstrates the use of an overridden method.

```java
public class Thought{
    
    public void message(){
        System.out.println("I feel like I'm diagonally parked in a " + "parallel universe.");
    }
}
```

A Advice class extends Thought and demonstrates how to override the parent's method.

```java
public class Advice extends Thought {
    
    public void message(){
        System.out.println("Warning: Dates in calendar are closer" + " than they appear.");
        //explicitly invokes the parent's version
        super.message();
    }
}
```
A child class can override the definition of an inherited method in favor of its own.

A method in the parent class can be invoked explicitly using the super reference, as in:

```java
super.message();
```

If a method is declared with the `final` modifier, it cannot be overridden.

The concept of overriding can be applied to variables and is called *shadowing variables*.

Shadowing variables should be avoided because it tends to cause unnecessarily confusing code.

---

- **Overloading** deals with multiple methods with the same name in the same class, but with different signatures.

- **Overriding** deals with two methods, one in a parent class and one in a child class, that have the same signature.

- Overloading lets you define a similar operation in different ways for different parameters.

- Overriding lets you define a similar operation in different ways for different object types.

---

- A child class can be the parent of another child, forming a class hierarchy.

- Two children of the same parent are called siblings.

- Common features should be put as high in the hierarchy as is reasonable.

- An inherited member is passed continually down the line.
  - Therefore, a child class inherits from all its ancestor classes.
There is no single class hierarchy that is appropriate for all situations

- A class called Object is defined in the java.lang package of the Java standard class library
- All classes are derived from the Object class
- If a class is not explicitly defined to be the child of an existing class, it is assumed to be the child of the Object class
- Therefore, the Object class is the ultimate root of all class hierarchies

- The Object class contains a few useful methods, which are inherited by all classes
  - I.e., the tostring() method is defined in the Object class
  - Every time we define the tostring method, we are actually overriding an inherited definition
  - The tostring method in the Object class is defined to return a string that contains the name of the objects class along with some other information
  - Also in Object:
    - equals() returns T if and only if 
    - clone() returns

Abstract classes and methods are shown in italics font.
An abstract class is a placeholder in a class hierarchy that represents a generic concept.

An abstract class cannot be instantiated.

To declare a class as abstract:

```java
public abstract class Shape {
    // contents
}
```

Abstract classes are an important element of software design: they allow us to establish common elements in a hierarchy that are too generic to instantiate.

An abstract class often contains abstract methods with no definitions.

- The abstract modifier must be applied to each abstract method.

An abstract class typically contains non-abstract methods with full definitions.

- A class declared as abstract does not have to contain abstract methods – simply declaring it as abstract makes it so.

The child of an abstract class must override the abstract methods of the parent, or it, too, will be considered abstract.

An abstract method cannot be defined as final or static.