8.1 – Inheritance

- 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).
- A programmer can tailor a derived class as needed by adding new variables or methods, or by modifying the inherited ones.
- The keyword extends is used to establish an is-a (inheritance) relationship.

```java
class Child extends Parent {
    // class contents
}
```

8.1 – Class Diagram for Words

8.1 – Words.java

```java
//********************************************************************
// Words.java       Java Foundations
// Demonstrates the use of an inherited method.
//********************************************************************
public class Words {
    public static void main (String[] args) {
        Dictionary webster = new Dictionary();
        System.out.println("Number of pages: "+ webster.getPages());
        System.out.println("Number of definitions: "+ webster.getDefinitions());
        System.out.println("Definitions per page: "+ webster.computeRatio());
    }
}
```

8.1 – Book.java

```java
//********************************************************************
// Book.java       Java Foundations
// Represents a book. Used as the parent of a derived class to demonstrate inheritance.
//********************************************************************
public class Book {
    protected int pages = 1500;
    // Note: No constructor
    // Pages setter (mutator).
    public void setPages (int numPages) {
        pages = numPages;
    }
    // Pages getter (accessor).
    public int getPages () {
        return pages;
    }
}
```
8.1 – Dictionary.java

```java
//********************************************************************
// Dictionary.java       Java Foundations
// Represents a dictionary, which is a book. Used to demonstrate inheritance.
//********************************************************************
public class Dictionary extends Book
{
    private int definitions = 52500;
    // Computes ratio of definitions per page
    public double computeRatio ()
    // to demo use of parent vars
    { return definitions/pages; }
    // Setter aka: mutator
    public void setDefinitions (int numDefinitions)
    { definitions = numDefinitions; }
    // Getter aka: accessor
    public int getDefinitions ()
    { return definitions; }
}
```

8.1 – The protected Modifier

- 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.
- Protected variables and methods can be shown with a # symbol preceding them in UML diagrams.

8.1 – Words2.java

```java
//********************************************************************
// Words2.java       Java Foundations
// Demonstrates the use of the super reference.
//********************************************************************
public class Words2
{
    // Instantiates a derived class and invokes its inherited and
    // local methods.
    public static void main (String[] args)
    { Dictionary2 webster = new Dictionary2 (1500, 52500);
        System.out.println (“Number of pages: “ + webster.getPages());
        System.out.println (“Number of definitions: “ + webster.getDefinitions());
        System.out.println (“Definitions per page: “ + webster.computeRatio());
    }
}
```
8.1 - Book2.java

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

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

    public void setPages(int numPages)
    {
        pages = numPages;
    }

    public int getPages()
    {
        return pages;
    }
}
```

8.1 - Dictionary2.java

```java
/**
 * Dictionary2.java       Java Foundations
 * Represents a dictionary, which is a book. Used to demonstrate
 * the use of the super reference.
 */
public class Dictionary2 extends Book2
{
    private int definitions;

    //-----------------------------------------------------------------
    // Constructor: Sets up the dictionary with the specified number
    // of pages and definitions.
    //-----------------------------------------------------------------
    public Dictionary2(int numPages, int numDefinitions)
    {
        super(numPages);
        definitions = numDefinitions;
    }

    (etc for computeRatio(), setDefinitions(), getDefinitions(…))
```

8.1 - Multiple Inheritance

- Java supports single inheritance, meaning that a derived class can have only one parent class.
- Multiple inheritance allows a class to be derived from two or more classes, inheriting the members of all parents.
- Collisions, such as the same variable name in two parents, have to be resolved.
- Java does not support multiple inheritance.
- C++ does.
- In most cases, the use of interfaces gives us aspects of multiple inheritance without the overhead.

8.2 - Overriding Methods

- A child class can override the definition of an inherited method in favor of its own.
- The overriding method must have the same signature as the parent’s, but can have a different body.
- The type of the object executing the method determines which version of the method is invoked.
### 8.2 - Messages.java

```java
public class Messages {
    public static void main (String[] args) {
        Thought parked = new Thought();
        Advice dates = new Advice();
        parked.message();
        dates.message(); // overridden
    }
}
```

### 8.2 - Thought.java

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

### 8.2 - Advice.java

```java
public class Advice extends Thought {
    public void message() {
        System.out.println("Warning: Dates in calendar are closer than they appear.");
        super.message(); // explicitly invokes the parent's version
    }
}
```

### 8.2 - Overriding

- 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.
**8.2 - Overloading vs. Overriding**

- **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.

**8.3 - Class Hierarchies**

- 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.

**8.3 - An Alternate Class Hierarchy**

There is no single class hierarchy that is appropriate for all situations.

**8.3 - The Object Class**

- 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.
8.3 - The Object Class’ Methods

- 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 object’s class along with some other information
- Also in `Object`:
  `equals()` returns `true` if __________
  `clone()` returns __________

8.3 - Abstract Classes

- 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:
  ```
  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

8.3 - Need for Abstract Classes

Abstract classes and
methods are shown in
italics font.

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