Overview of Java’s Support for Inheritance

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Lesson

• Understand what inheritance is & how it’s supported in Java
Overview of Java’s Support for Inheritance
Overview of Java’s Support for Inheritance

- OO languages enhance reuse by allowing classes to inherit commonly used state & behavior from other classes

See en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)
Inheritance in Java is specified via its `extends` keyword.

```
AbstractSet
int MAX_ARRAY_SIZE
...
boolean hashCode()
...

<<extends>>

SimpleSet
Object[] mElementData
...
boolean add(E e)
...
```

See [docs.oracle.com/javase/tutorial/java/IandI/subclasses.html](docs.oracle.com/javase/tutorial/java/IandI/subclasses.html)
Overview of Java’s Support for Inheritance

- Inheritance in Java is specified via its `extends` keyword
- Allows a subclass to inherit all non-private fields & methods from a super class

See [docs.oracle.com/javase/tutorial/java/IandI/subclasses.html](docs.oracle.com/javase/tutorial/java/IandI/subclasses.html)
Inheritance in Java is specified via its \texttt{extends} keyword.

- Allows a subclass to inherit all non-private fields & methods from a super class.

\begin{itemize}
  \item \texttt{SimpleSet} inherits \texttt{MAX\_ARRAY\_SIZE} & \texttt{hashCode()} (among others) from \texttt{AbstractSet}.
\end{itemize}
Inheritance in Java is specified via its `extends` keyword.

- Allows a subclass to inherit all non-private fields & methods from a super class.

```java
Set<String> s = new SimpleSet();
s.hashCode();
s.add(element);
```

**AbstractSet**
- `int MAX_ARRAY_SIZE`
- `boolean hashCode()`
- ...

**SimpleSet**
- `Object[] mElementData`
- `boolean add(E e)`
- ...

<<extends>>
Inheritance in Java is specified via its `extends` keyword.

- Allows a subclass to inherit all non-private fields & methods from a super class.

```java
Set<String> s = new SimpleSet();
s.hashCode();
s.add(element);
```
Overview of Java’s Support for Inheritance

- Inheritance in Java is specified via its `extends` keyword
- Allows a subclass to inherit all non-private fields & methods from a super class

```java
Set<String> s =
    new SimpleSet();
s.hashCode();
s.add(element);
```
Inheritance in Java is specified via its `extends` keyword.

- Allows a subclass to inherit all non-private fields & methods from a super class.

```java
Set<String> s = new SimpleSet();
s.hashCode();
s.add(element);
```

SimpleSet can also “override” `hashCode()`, as discussed in *Polymorphism* lesson.
Overview of Java’s Support for Inheritance

- Java Collections Framework demonstrates capabilities & benefits of inheritance

See docs.oracle.com/javase/8/docs/technotes/guides/collections
Overview of Java’s Support for Inheritance

- Java Collections Framework demonstrates capabilities & benefits of inheritance

See the lesson on “Overview of the Java Collections Framework”
Overview of Java’s Support for Inheritance

- The List hierarchy are collections that maintain an ordering for their elements.

See [docs.oracle.com/javase/8/docs/api/java/util/List.html](docs.oracle.com/javase/8/docs/api/java/util/List.html)
Overview of Java’s Support for Inheritance

- The List hierarchy are collections that maintain an ordering for their elements.

See docs.oracle.com/javase/8/docs/api/java/util/List.html
Overview of Java’s Support for Inheritance

- The List hierarchy are collections that maintain an ordering for their elements

See docs.oracle.com/javase/8/docs/api/java/util/AbstractList.html
Overview of Java’s Support for Inheritance

- The List hierarchy are collections that maintain an ordering for their elements.


See [docs.oracle.com/javase/8/docs/api/java/util/Vector.html](https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html)
Overview of Java’s Support for Inheritance

- The List hierarchy are collections that maintain an ordering for their elements.
Overview of Java’s Support for Inheritance

- The List hierarchy are collections that maintain an ordering for their elements.

This inheritance hierarchy enhances systematic reuse of data fields & methods.
The Role of the Java Object Super Class
The Role of the Java Object Super Class

- All Java classes inherit from the java.lang.Object super class

```java
package java.lang;

public class Object {
    ...
    public int hashCode();
    public boolean equals (Object o);
    ...
    public final void wait();
    public final void notify();
    public final void notifyAll();
    ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/lang/Object.html](docs.oracle.com/javase/8/docs/api/java/lang/Object.html)
All Java classes inherit from the java.lang.Object super class

Defines methods that can be used by all non-primitive types

package java.lang;

public class Object {
    ...
    public int hashCode();
    public boolean equals (Object o);
    ...
    public final void wait();
    public final void notify();
    public final void notifyAll();
    ...

See docs.oracle.com/javase/8/docs/api/java/lang/Object.html
The Role of the Java Object Super Class

- All Java classes inherit from the `java.lang.Object` super class
- Defines methods that can be used by all non-primitive types

```java
package java.lang;

public class Object {
    ...
    public int hashCode();
    public boolean equals (Object o);
    ...
    public final void wait();
    public final void notify();
    public final void notifyAll();
    ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/lang/Object.html#hashCode](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#hashCode)
The Role of the Java Object Super Class

- All Java classes inherit from the java.lang.Object super class
- Defines methods that can be used by all non-primitive types

```java
package java.lang;

public class Object {
    ...
    public int hashCode();
    public boolean equals (Object o);
    ...
    public final void wait();
    public final void notify();
    public final void notifyAll();
    ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals](docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals)
The Role of the Java Object Super Class

- All Java classes inherit from the java.lang.Object super class
- Defines methods that can be used by all non-primitive types

```java
package java.lang;

public class Object {
    ...
    public int hashcode();
    public boolean equals(Object o);
    ...
    public final void wait();
    public final void notify();
    public final void notifyAll();
    ...

See docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html
The Role of the Java Object Super Class

• Subclasses that don’t explicitly extend a super class implicitly inherit from java.lang.Object

```java
package java.lang;

public abstract class Process {
    ...
    public abstract int waitFor();
    ...
    ...
```
Subclasses that don’t explicitly extend a super class implicitly inherit from java.lang.Object, e.g.

- java.lang.Process implicitly extends java.lang.Object

```java
package java.lang;

public abstract class Process {
    ...
    public abstract int waitFor();
    ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/lang/Process.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Process.html)
Subclasses that don’t explicitly extend a super class implicitly inherit from java.lang.Object, e.g.

- java.lang.Process implicitly extends java.lang.Object

All instances of java.lang.Process therefore also provide clients access to inherited java.lang.Object methods

- All objects, including arrays, implement the methods of this class
The Role of the Java Object Super Class

• `java.lang.Object` is the most general of all classes
java.lang.Object is the most general of all classes
It serves as the root of a hierarchy of classes available to Java apps
The Role of the Java Object Super Class

• `java.lang.Object` is the most general of all classes
  • It serves as the root of a hierarchy of classes available to Java apps
• Classes towards the bottom of the inheritance hierarchy are more specialized
java.lang.Object is the most general of all classes

- It serves as the root of a hierarchy of classes available to Java apps

- Classes towards the bottom of the inheritance hierarchy are more specialized
  - e.g., List-related subclasses override methods inherited from super classes
The Role of the Java Object Super Class

- `java.lang.Object` defines `equals()`
- If operator `==` is used to compare the equality of two objects it returns true if the two objects have the same memory address
The Role of the Java Object Super Class

- `java.lang.Object` defines `.equals()`
  - If `operator==` is used to compare the equality of two objects it returns true if the two objects have the same memory address
  
- Conversely, if you use `.equals()` to compare for equality, a subclass can override this method to do other things
  - e.g., check for equal values in a collection or string

```
Object
wait()
notify();
notifyAll();
equals();
...
Process
waitFor()
...
```
The Three Purposes of Subclass Methods
The Three Purposes of Subclass Methods

- Subclass methods inherited from a super class are used for 3 purposes

```java
public class Stack<E> {
    extends Vector<E> {
```
The Three Purposes of Subclass Methods

- Subclass methods inherited from a super class are used for 3 purposes

```java
public class Stack<E> {
    extends Vector<E> {
```

Extends Vector to define a last-in/first-out data structure that enables apps to pop & push items to/from a stack

See [docs.oracle.com/javase/8/docs/api/java/util/Stack.html](https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html)
The Three Purposes of Subclass Methods

- Subclass methods inherited from a super class are used for 3 purposes

```java
public class Stack<E> {
    extends Vector<E> {
```

Extends Vector to define a last-in/first-out data structure that enables apps to pop & push items to/from a stack

See [docs.oracle.com/javase/8/docs/api/java/util/Vector.html](https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html)
The Three Purposes of Subclass Methods

• Subclass methods inherited from a super class are used for 3 purposes

1. To augment the subclass API

public class Stack<E> {
    extends Vector<E> {


The Three Purposes of Subclass Methods

• Subclass methods inherited from a super class are used for 3 purposes

1. To augment the subclass API
   • e.g., Stack subclass inherits fields & methods from Vector super class

```java
public class Stack<E> {
    extends Vector<E> {
```
The Three Purposes of Subclass Methods

- Subclass methods inherited from a super class are used for 3 purposes

1. To augment the subclass API
   - e.g., Stack subclass inherits fields & methods from Vector super class

   ```java
   public class Stack<E> {
       extends Vector<E> {

           e.g.,
           Stack<Integer> s =
               new Stack<>();
           ...
           if(!s.isEmpty())
               s.pop();
   }
   ```

   `isEmpty()` method inherited from Vector can be invoked on a Stack instance

This Stack class uses so-called “implementation inheritance”
The Three Purposes of Subclass Methods

- Subclass methods inherited from a super class are used for 3 purposes
  1. To augment the subclass API
     - e.g., Stack subclass inherits fields & methods from Vector super class

```
public class Stack<E> {
    extends Vector<E> {

    
    
e.g.,

    Stack<Integer> s =
        new Stack<>();

    ...

    if(!s.isEmpty())
        s.pop();
```

A method can be invoked on an object instance
The Three Purposes of Subclass Methods

• Subclass methods inherited from a super class are used for 3 purposes

1. To augment the subclass API
2. To implement subclass methods
3. To implement inheritance methods like addElement()

Due to implementation inheritance methods like addElement() are visible to Stack clients.
The Three Purposes of Subclass Methods

- Subclass methods inherited from a super class are used for 3 purposes
- 1. To augment the subclass API
- 2. To implement subclass methods
  - A subclass may add new methods & data members

```java
public class Stack<E> {
    extends Vector<E> {
        ...
        public Object push(E e) {
            addElement(e);
            return e;
        }
        ...
    }
}
```

The Stack's `push()` method is implemented via Vector's `addElement()` method.
Subclass methods inherited from a super class are used for 3 purposes:

1. To augment the subclass API
2. To implement subclass methods
3. To override super class methods in subclass with same signatures

Example:

```java
public abstract class AbstractMap<K,V> ...

public abstract Set<Entry<K,V>> entrySet();
...

public HashMap<K,V> extends AbstractMap<K,V> ...

public Set<Entry<K,V>> entrySet() {
    ... }
...
```

HashMap’s `entrySet()` method overrides AbstractMap’s `entrySet()` abstract method.

See [en.wikipedia.org/wiki/Type_signature#Java_2](en.wikipedia.org/wiki/Type_signature#Java_2)
The Three Purposes of Subclass Methods

- Subclass methods inherited from a super class are used for 3 purposes
  1. To augment the subclass API
  2. To implement subclass methods
  3. To override super class methods in subclass with same signatures

  • A subclass may define methods with the same signatures as super class methods

```
public abstract class AbstractMap<K,V> ...

public abstract
Set<Entry<K,V>> entrySet(); ...
```

```
public HashMap<K,V> extends AbstractMap<K,V> ...

public Set<Entry<K,V>>
entrySet() { ... } ...
```

HashMap’s entrySet() method overrides AbstractMap’s entrySet() abstract method

See [docs.oracle.com/javase/tutorial/java/IandI/override.html](docs.oracle.com/javase/tutorial/java/IandI/override.html)
Subclass methods inherited from a super class are used for 3 purposes

1. To augment the subclass API
2. To implement subclass methods
3. To override super class methods in subclass with same signatures

- A subclass may define methods with the same signatures as super class methods
- These methods “override” the original methods

The Three Purposes of Subclass Methods

public abstract class AbstractMap<K,V> ...
  public abstract Set<Entry<K,V>> entrySet();
...

HashMap's entrySet() method overrides AbstractMap's entrySet() abstract method

public HashMap<K,V> extends AbstractMap<K,V> ...
  public Set<Entry<K,V>>
    entrySet() { ... }
  ...

See en.wikipedia.org/wiki/Method_overriding
The Three Purposes of Subclass Methods

- Subclass methods inherited from a superclass are used for 3 purposes:
  1. To augment the subclass API
  2. To implement subclass methods
  3. To override superclass methods in subclass with same signatures

- A subclass may define methods with the same signatures as superclass methods
- These methods “override” the original methods

Method overriding is covered in the next lesson on *Java Polymorphism*
End of Overview of Java’s Support for Inheritance