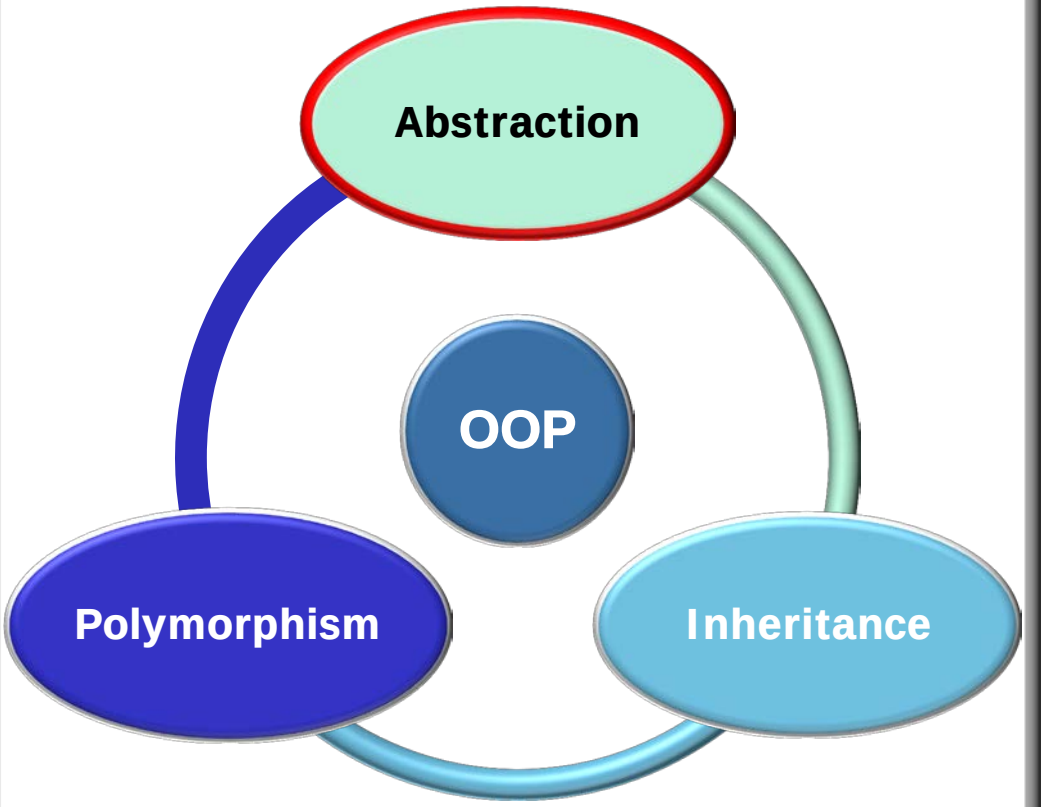
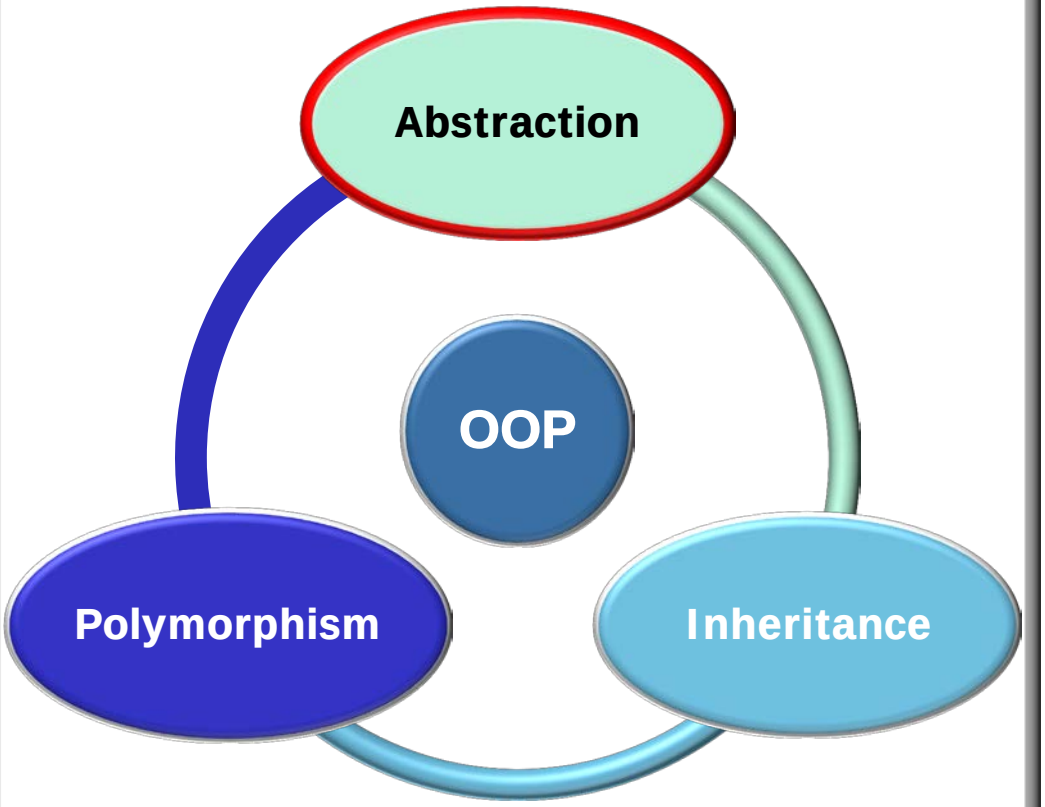
Overview of the Java Programming Language

Overview of the Java Programming Language



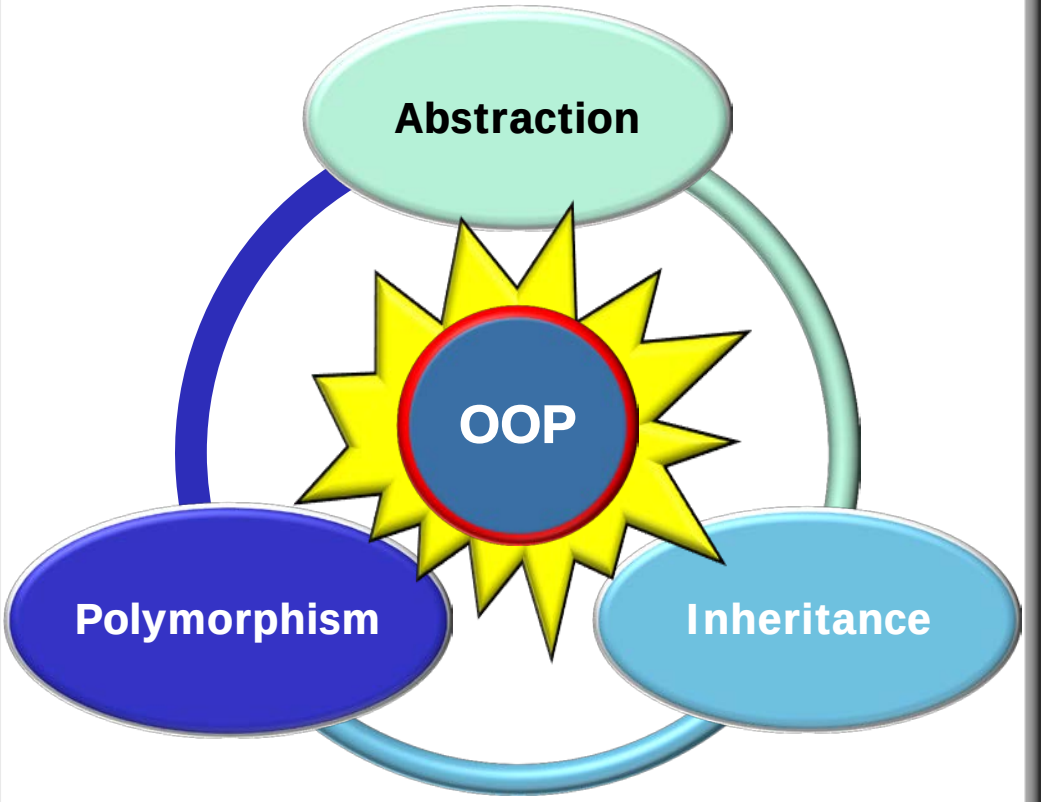
See www.stroustrup.com/whatis.pdf

Overview of the Java Programming Language



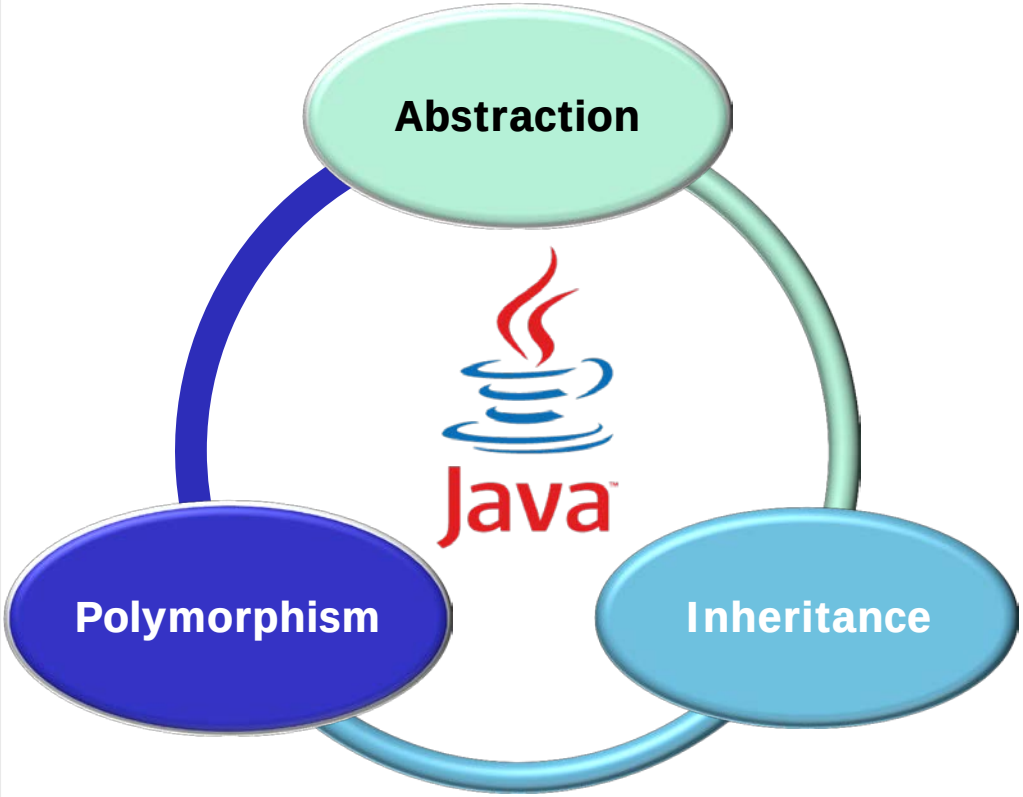
Abstraction makes it easier to develop robust apps that can evolve flexibly

Overview of the Java Programming Language



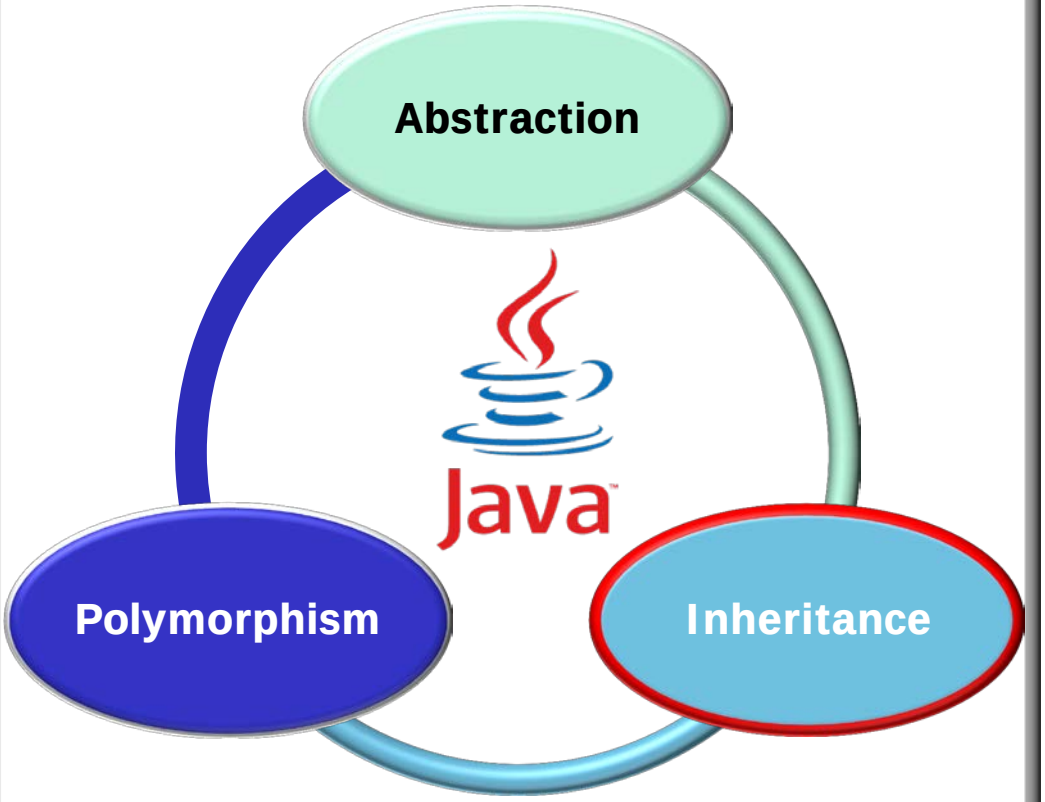
See www.stroustrup.com/whatis.pdf

Overview of the Java Programming Language



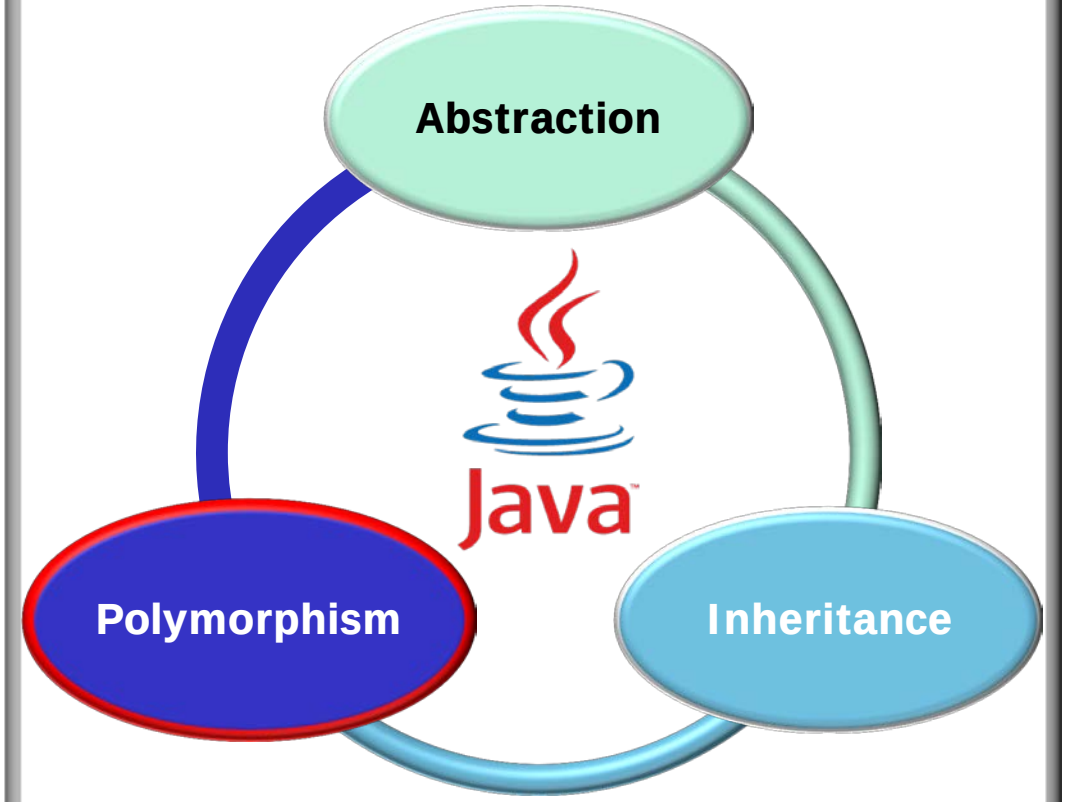
See www.stroustrup.com/whatis.pdf

Overview of the Java Programming Language



See www.stroustrup.com/whatis.pdf

Overview of the Java Programming Language



See www.stroustrup.com/whatis.pdf

Overview of the Java Programming Language

Learning Objectives

- Understand what these advanced object-oriented (OO) concepts mean
- Know the benefits they provide developers of Java apps in Android
- Identify Java features that implement these OO concepts

Overview of the Java Programming Language

Learning Objectives

- Understand what these advanced object-oriented (OO) concepts mean
- Know the benefits they provide developers of Java apps in Android
- Identify Java features that implement these OO concepts

Overview of the Java Programming Language

Learning Objectives

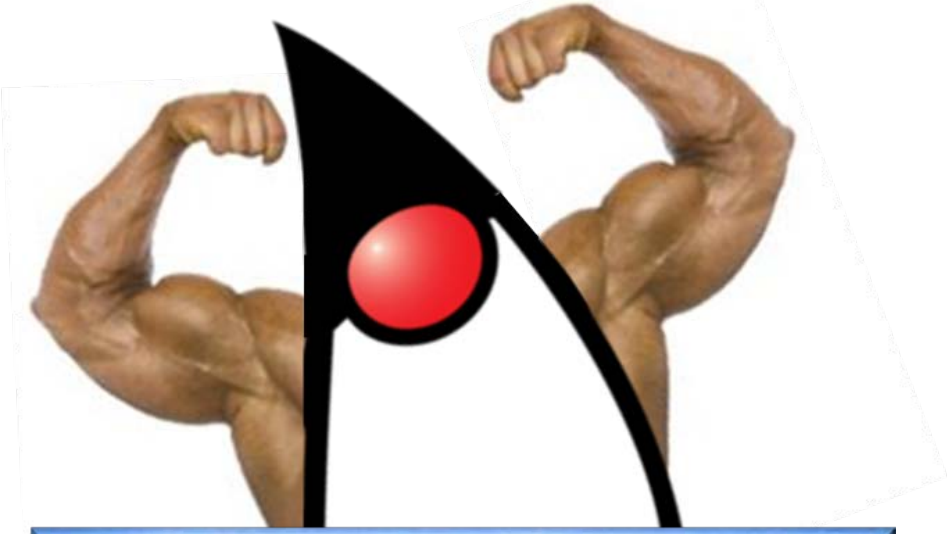
- Understand what these advanced object-oriented (OO) concepts mean
- Know the benefits they provide developers of Java apps in Android
- Identify Java features that implement these OO concepts

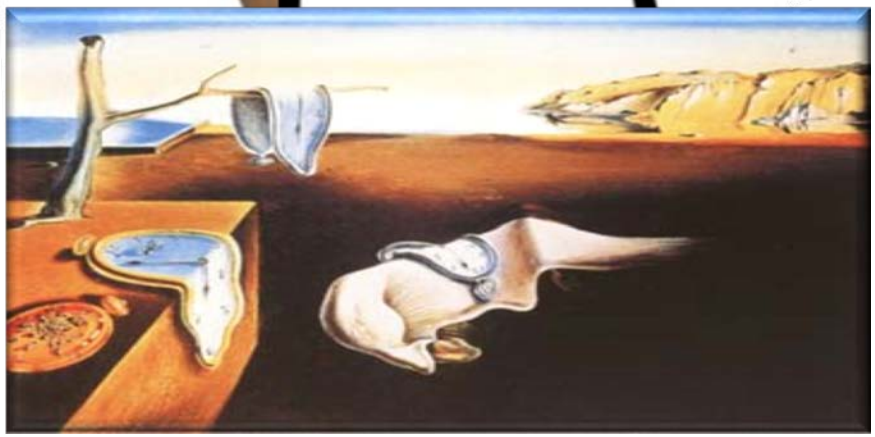
Overview of the Java Programming Language

Learning Objectives

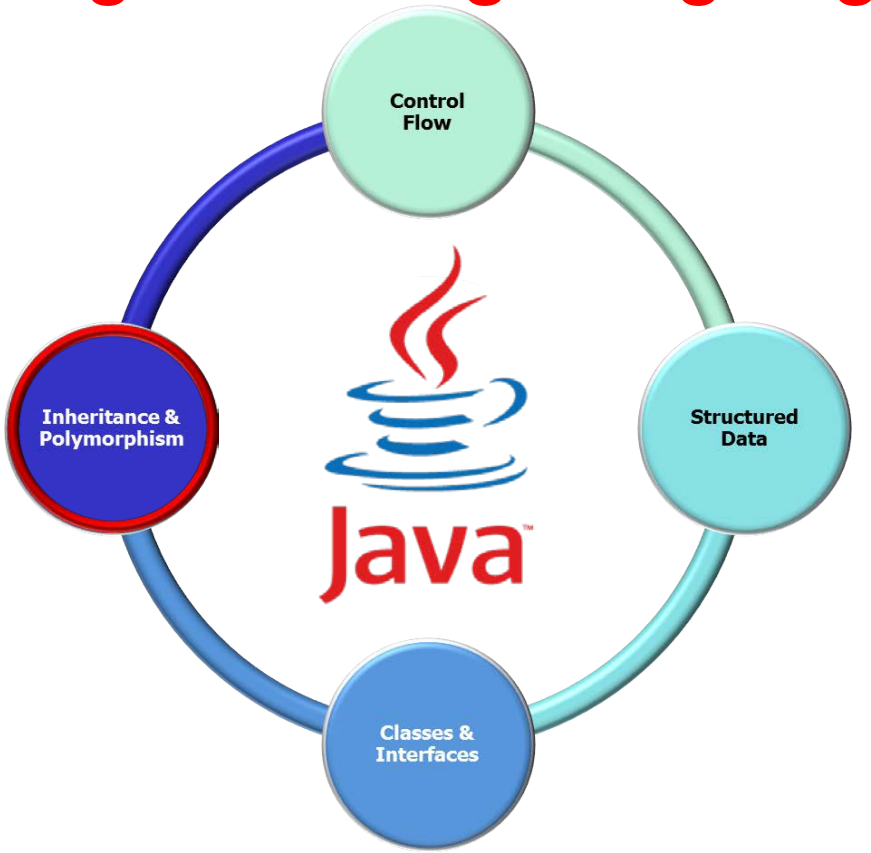
- Understand what these advanced object-oriented (OO) concepts mean
- Know the benefits they provide developers of Java apps in Android
- Identify Java features that implement these OO concepts

Overview of the Java Programming Language





Overview of the Java Programming Language

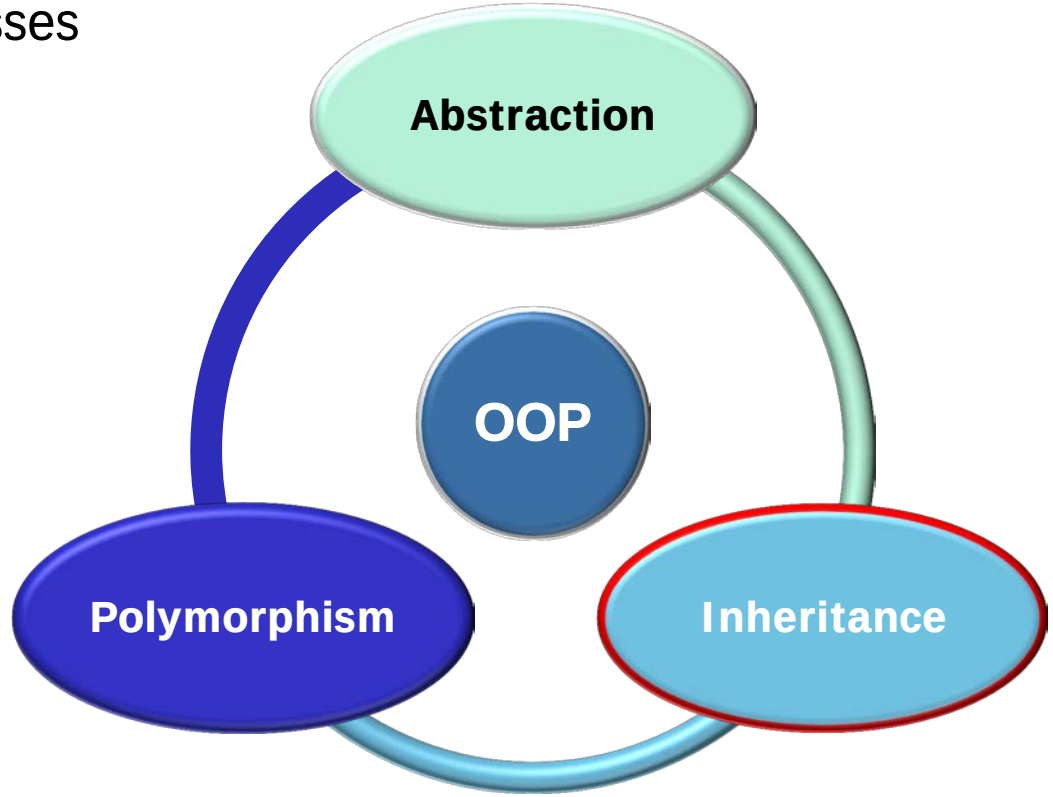


Other lessons examine Java inheritance & polymorphism in detail

Overview of Java's Support for Inheritance (Part 1)

Overview of Java's Support for Inheritance

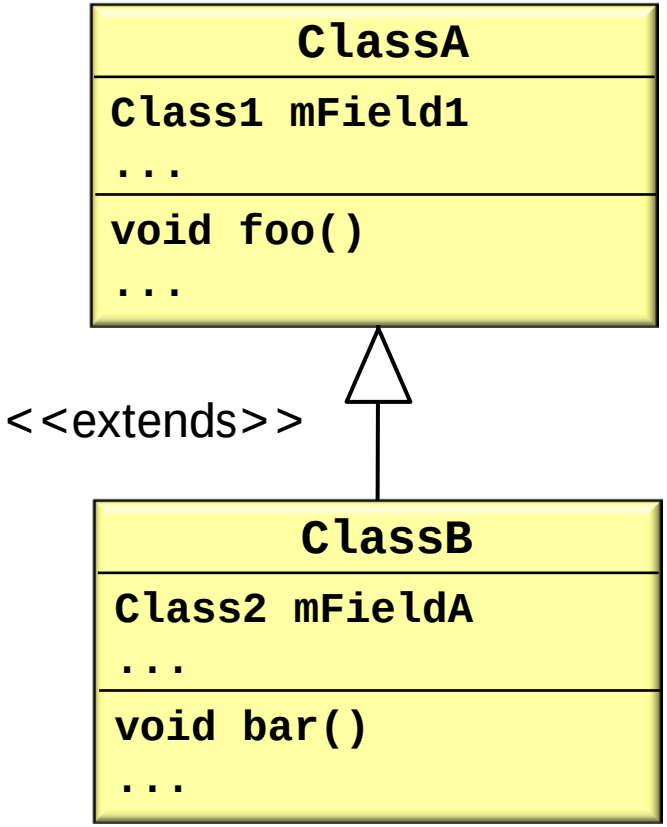
- OO languages enhance reuse by allowing classes to inherit commonly used state & behavior from other classes



See en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)

Overview of Java's Support for Inheritance

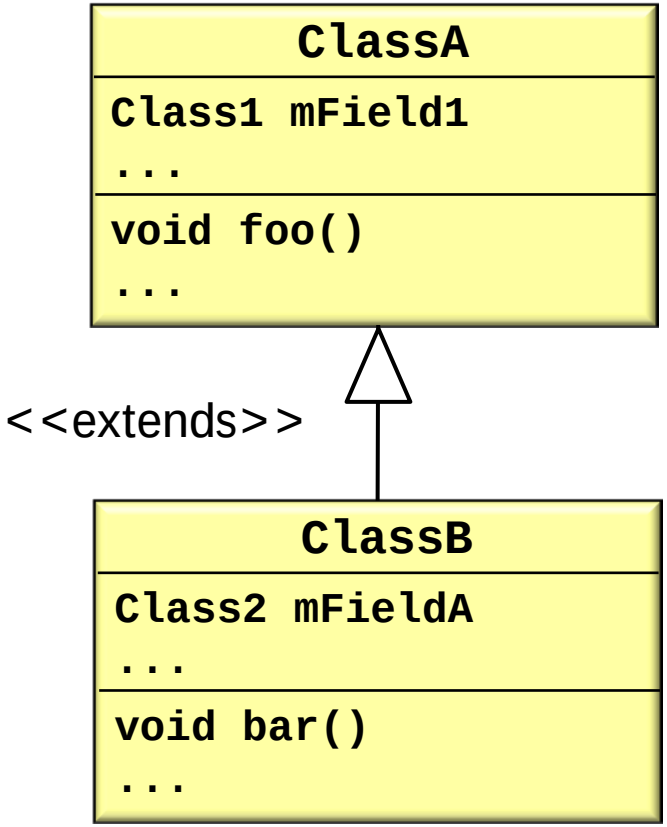
- Inheritance in Java is specified via its **`extends`** keyword



See docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Overview of Java's Support for Inheritance

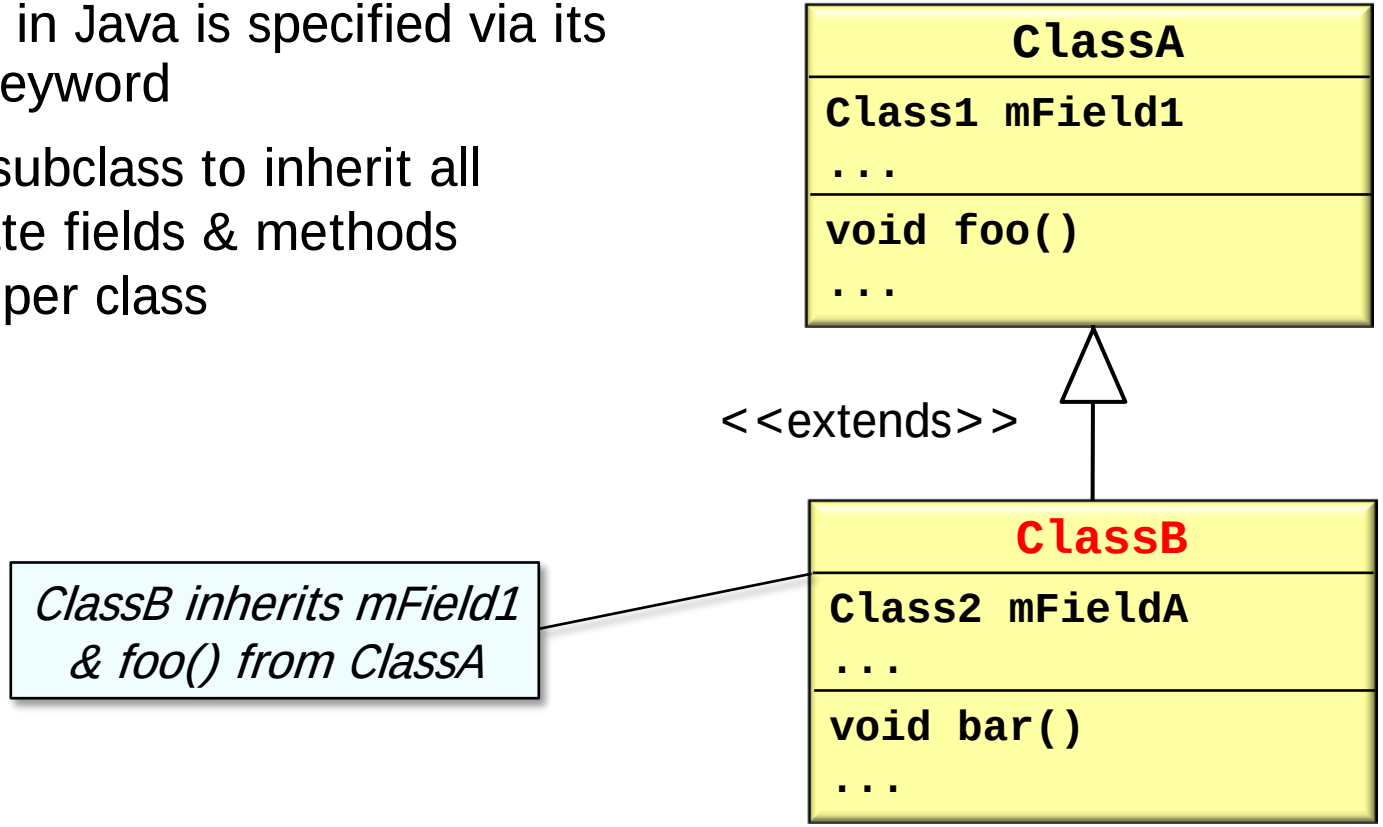
- Inheritance in Java is specified via its **`extends`** keyword
  - Allows a subclass to inherit all non-private fields & methods from a super class



See docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

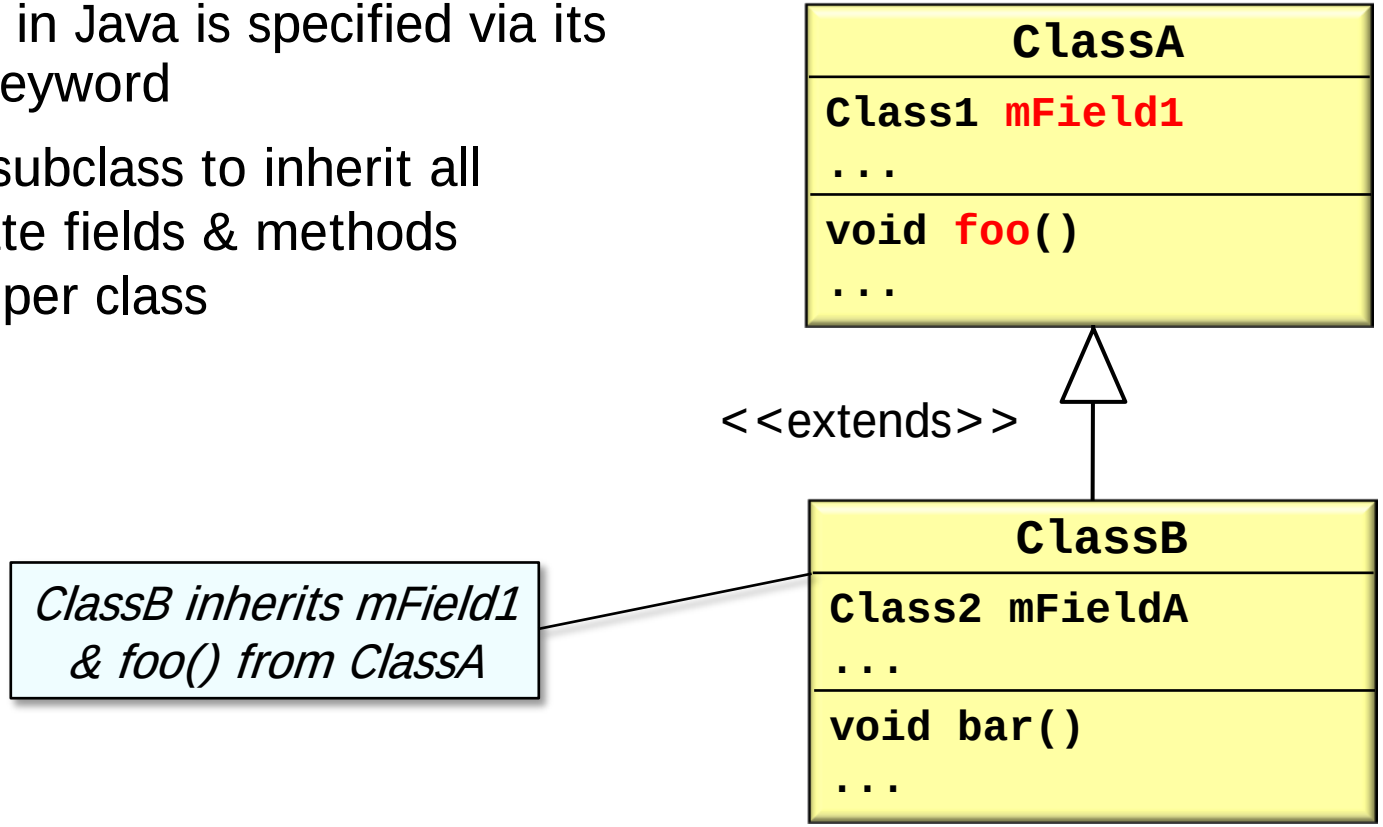
Overview of Java's Support for Inheritance

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



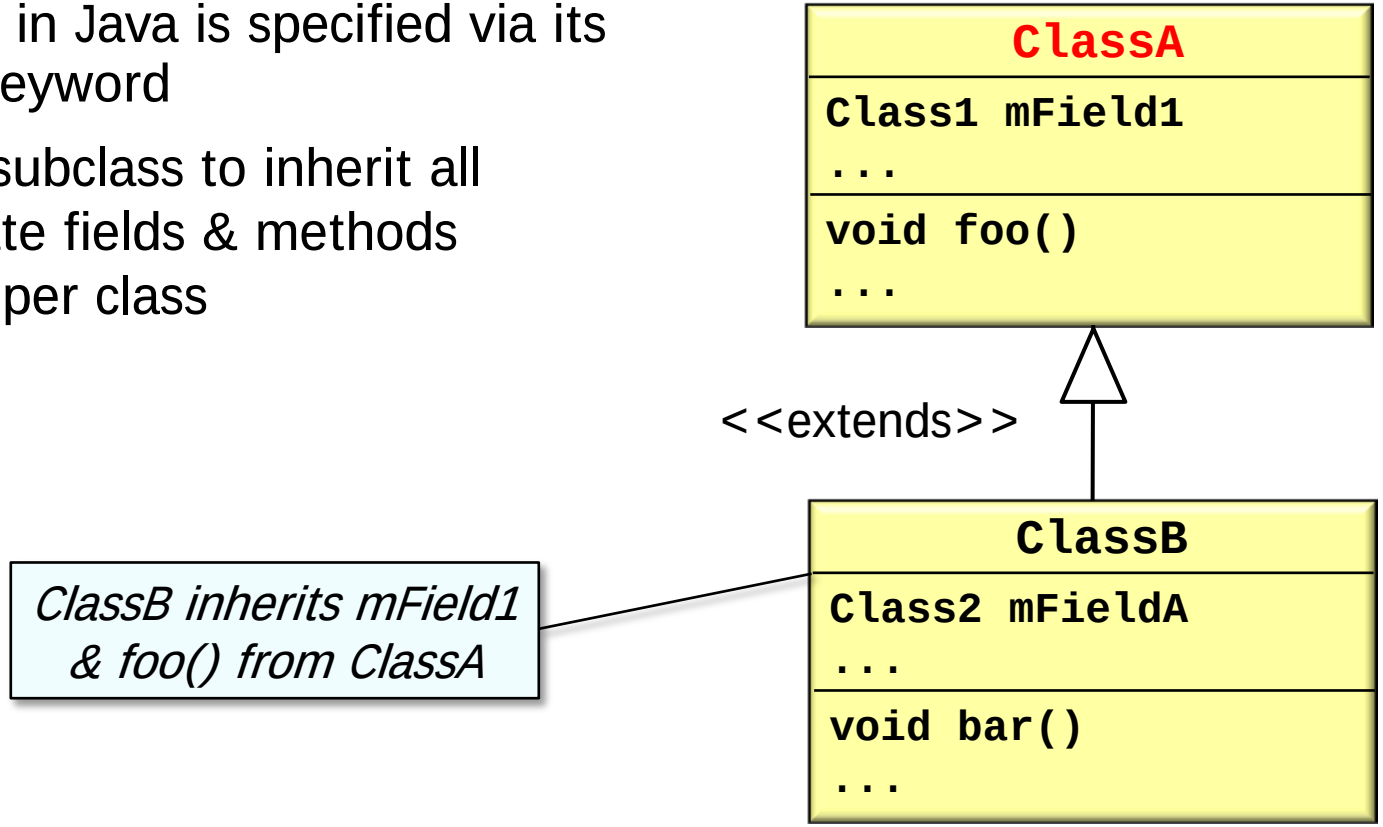
Overview of Java's Support for Inheritance

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



Overview of Java's Support for Inheritance

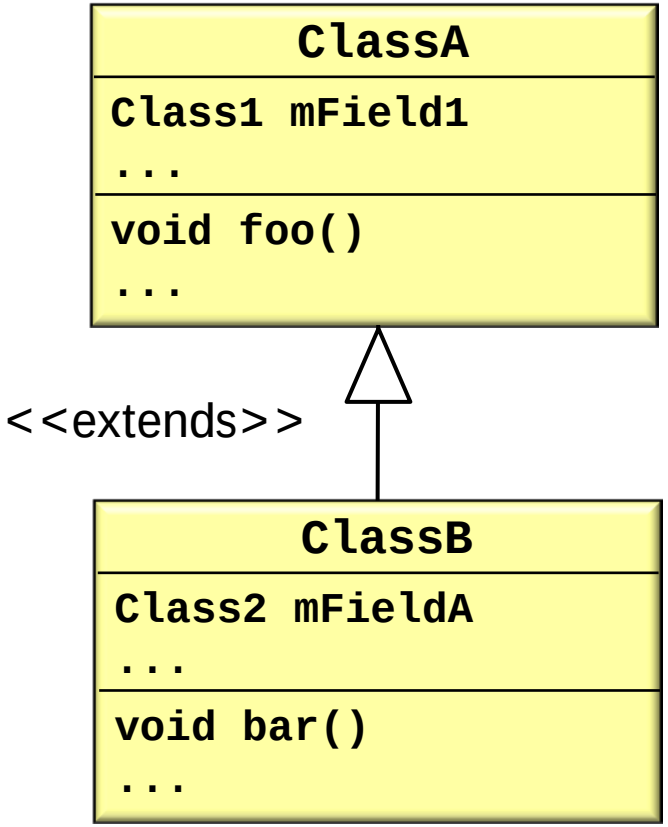
- Inheritance in Java is specified via its **`extends`** keyword
  - Allows a subclass to inherit all non-private fields & methods from a super class



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

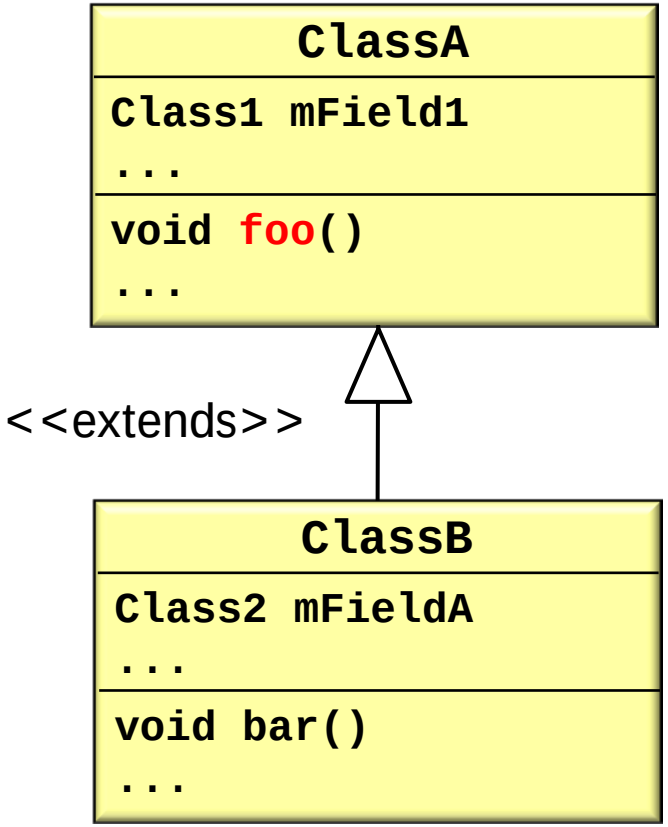
```
Class B b = new ClassB();
b.foo();
b.bar();
```



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

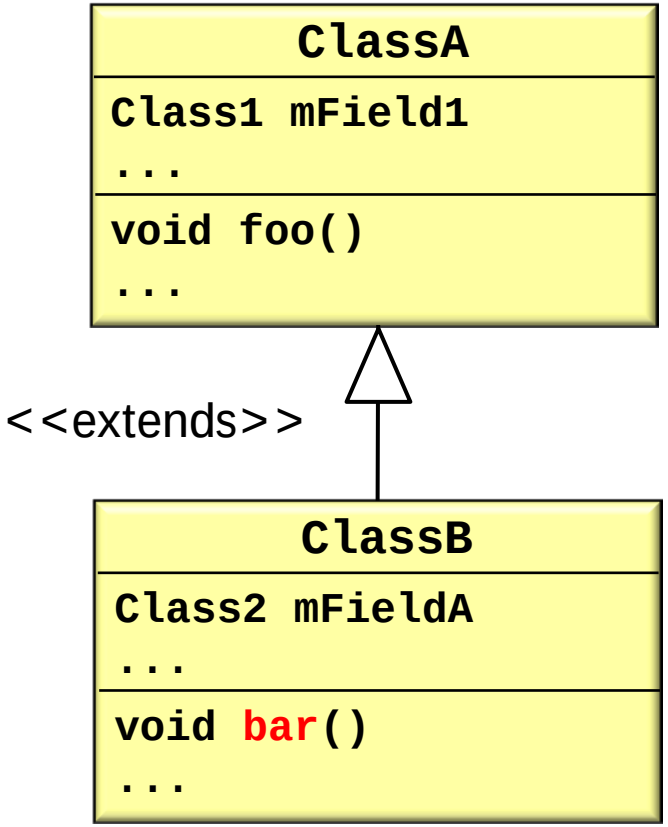
```
Class B b = new ClassB();
b.foo();
b.bar();
```



Overview of Java's Support for Inheritance

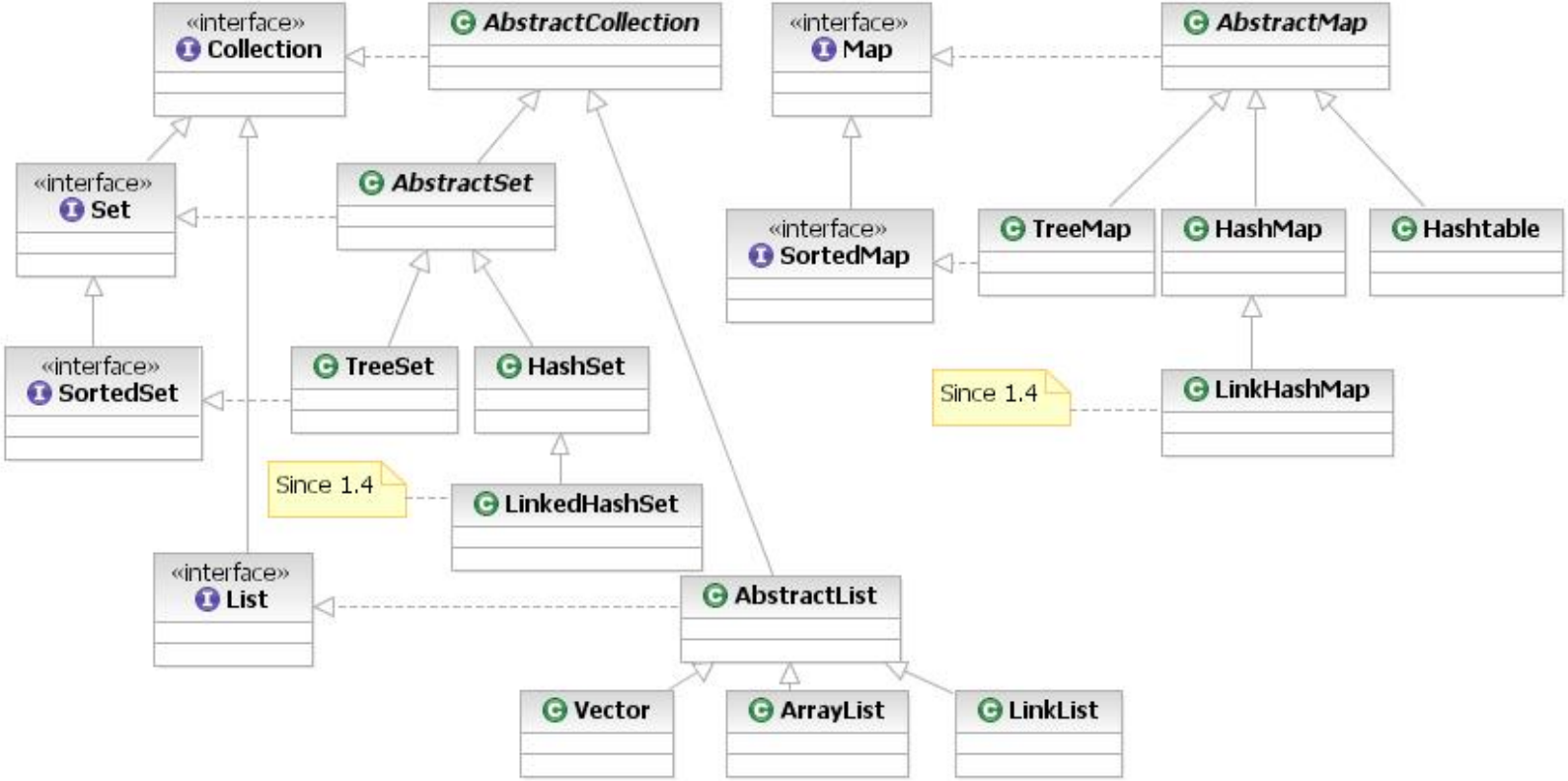
- Inheritance in Java is specified via its **`extends`** keyword
  - Allows a subclass to inherit all non-private fields & methods from a super class

```
Class B b = new ClassB();
b.foo();
b.bar();
```



Overview of Java's Support for Inheritance

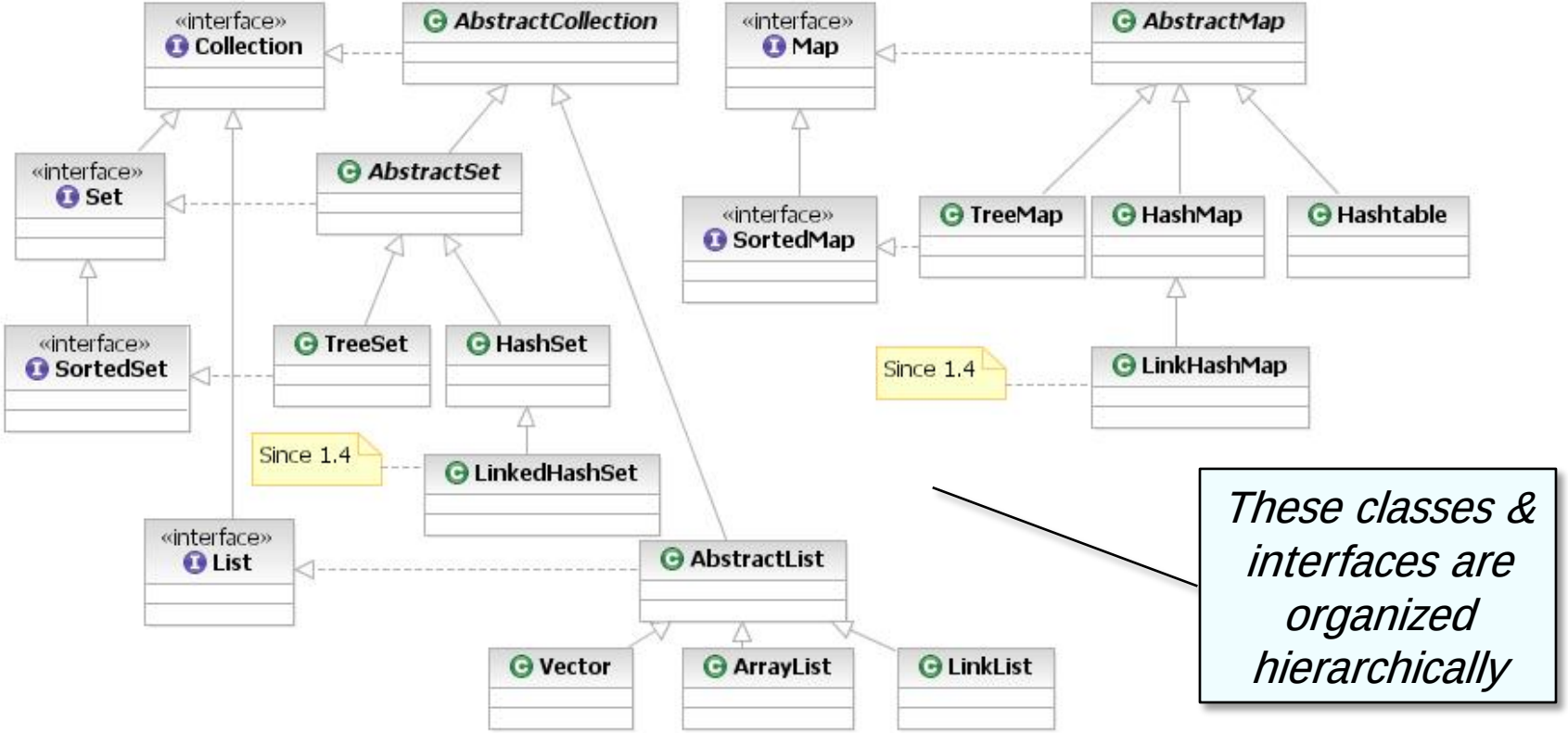
- Java Collections Framework demonstrates capabilities & benefits of inheritance



See docs.oracle.com/javase/8/docs/technotes/guides/collections

Overview of Java's Support for Inheritance

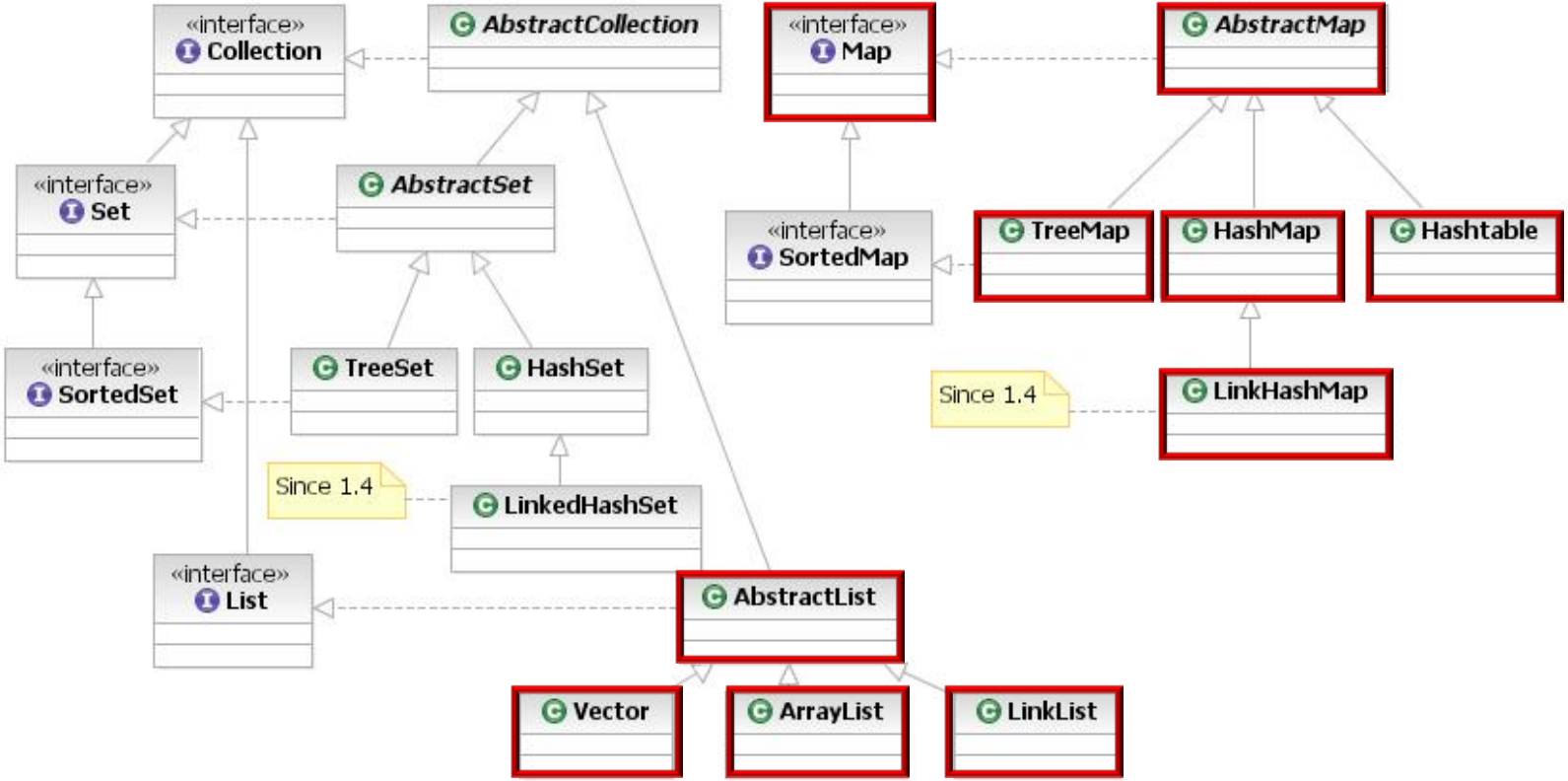
- Java Collections Framework demonstrates capabilities & benefits of inheritance



See docs.oracle.com/javase/8/docs/technotes/guides/collections

Overview of Java's Support for Inheritance

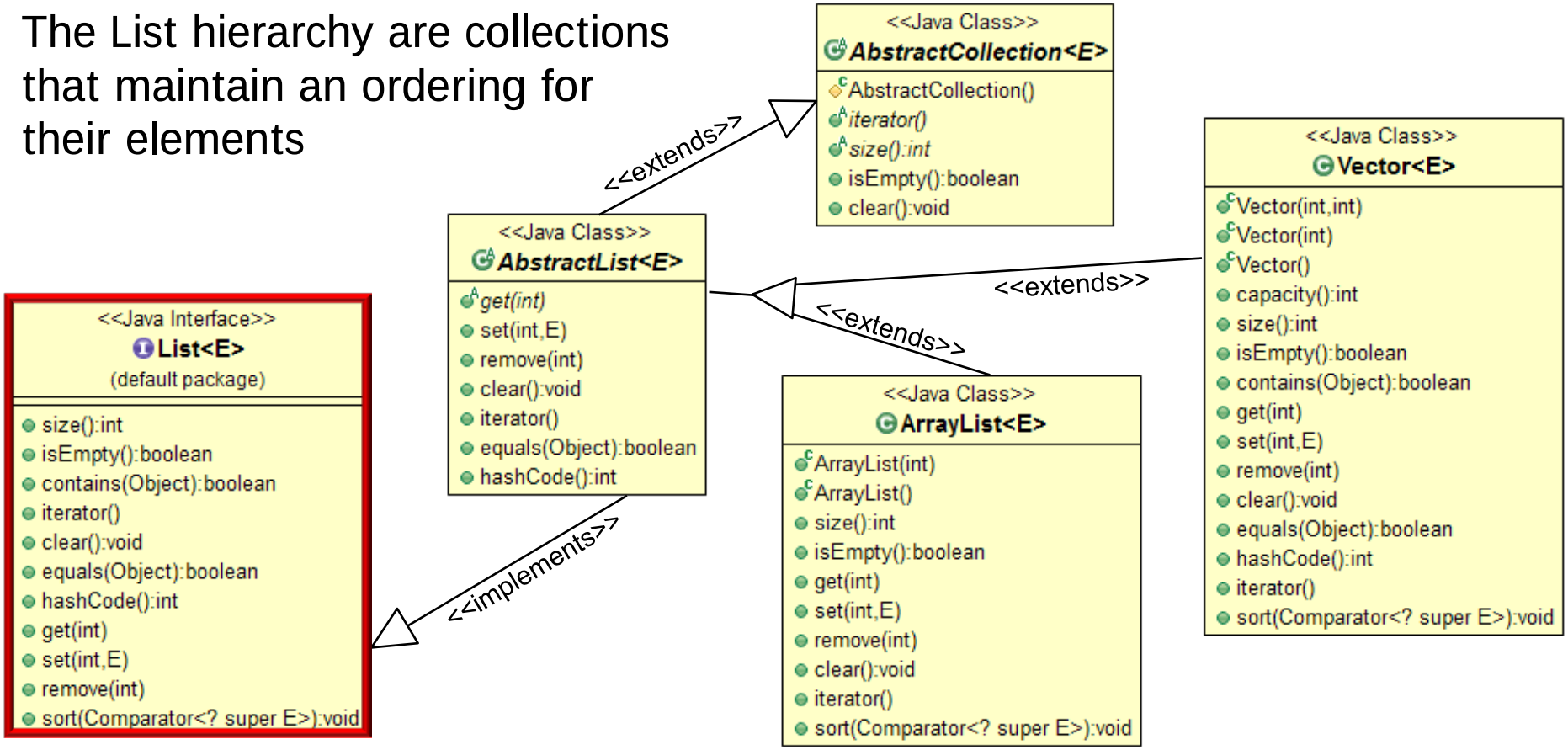
- Java Collections Framework demonstrates capabilities & benefits of inheritance



See the module on "Structured Data" for more on the Java Collections Framework

Overview of Java's Support for Inheritance

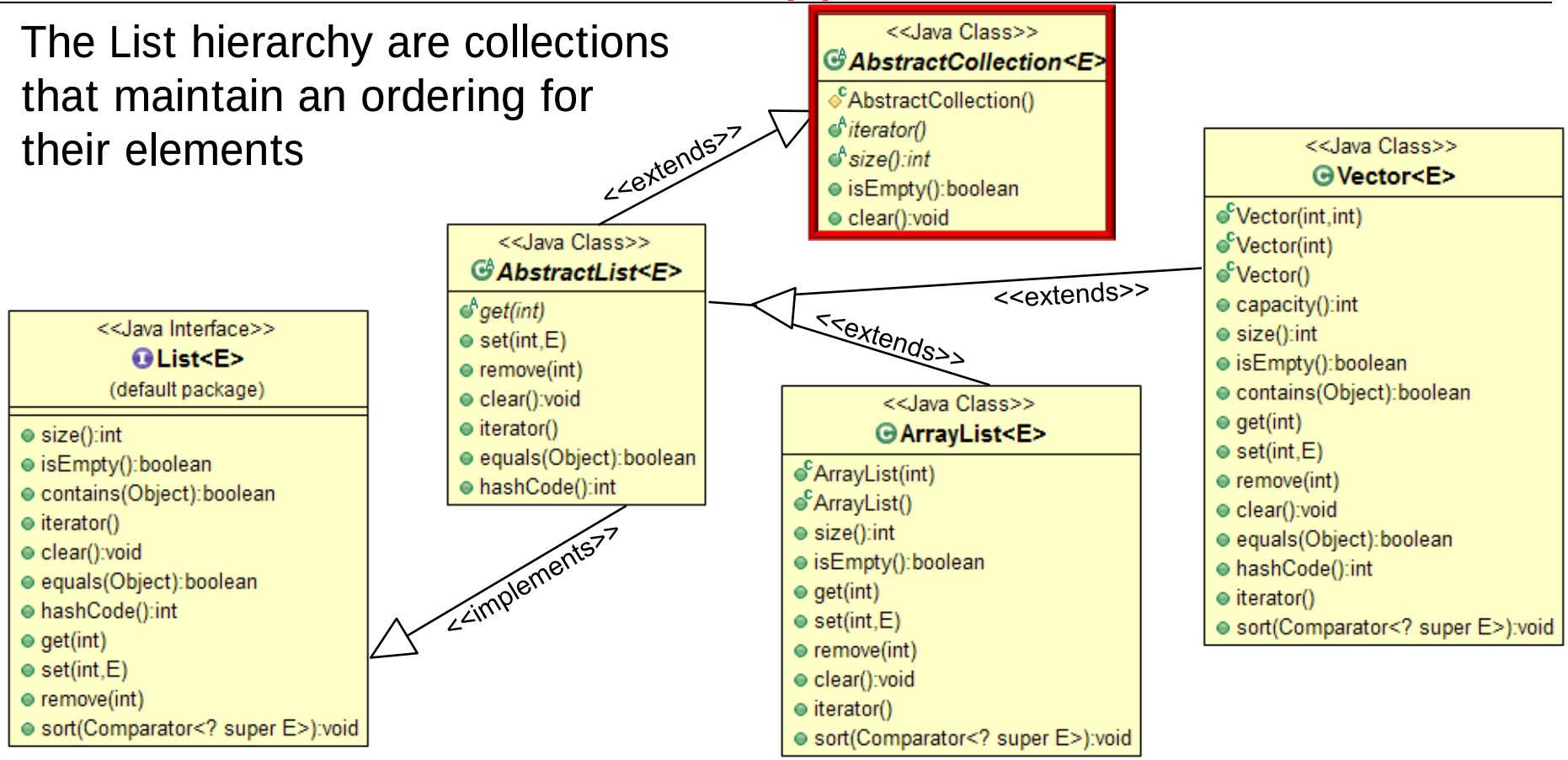
- The List hierarchy are collections that maintain an ordering for their elements



See developer.android.com/reference/java/util/List.html

Overview of Java's Support for Inheritance

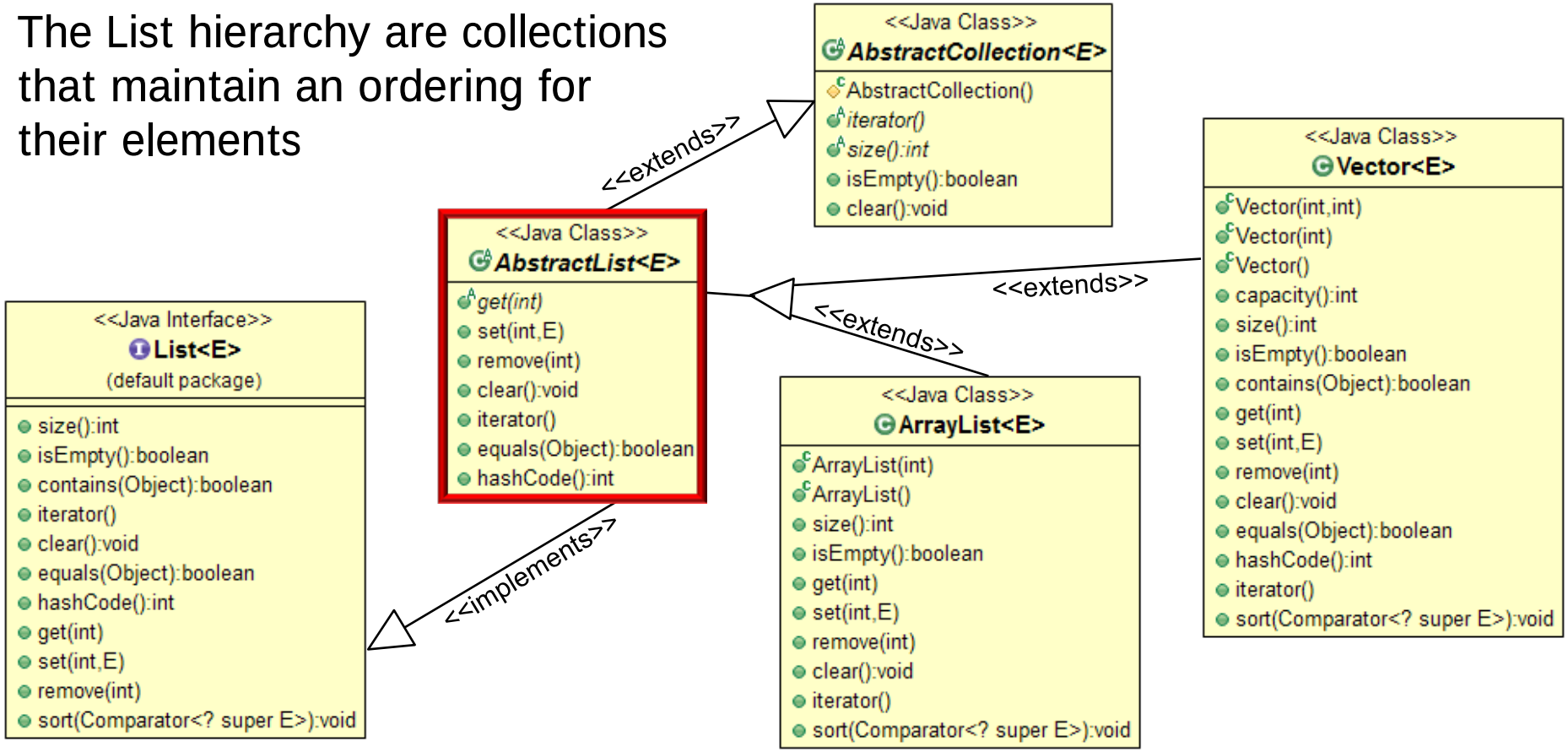
- The List hierarchy are collections that maintain an ordering for their elements



See developer.android.com/reference/java/util/AbstractCollection.html

Overview of Java's Support for Inheritance

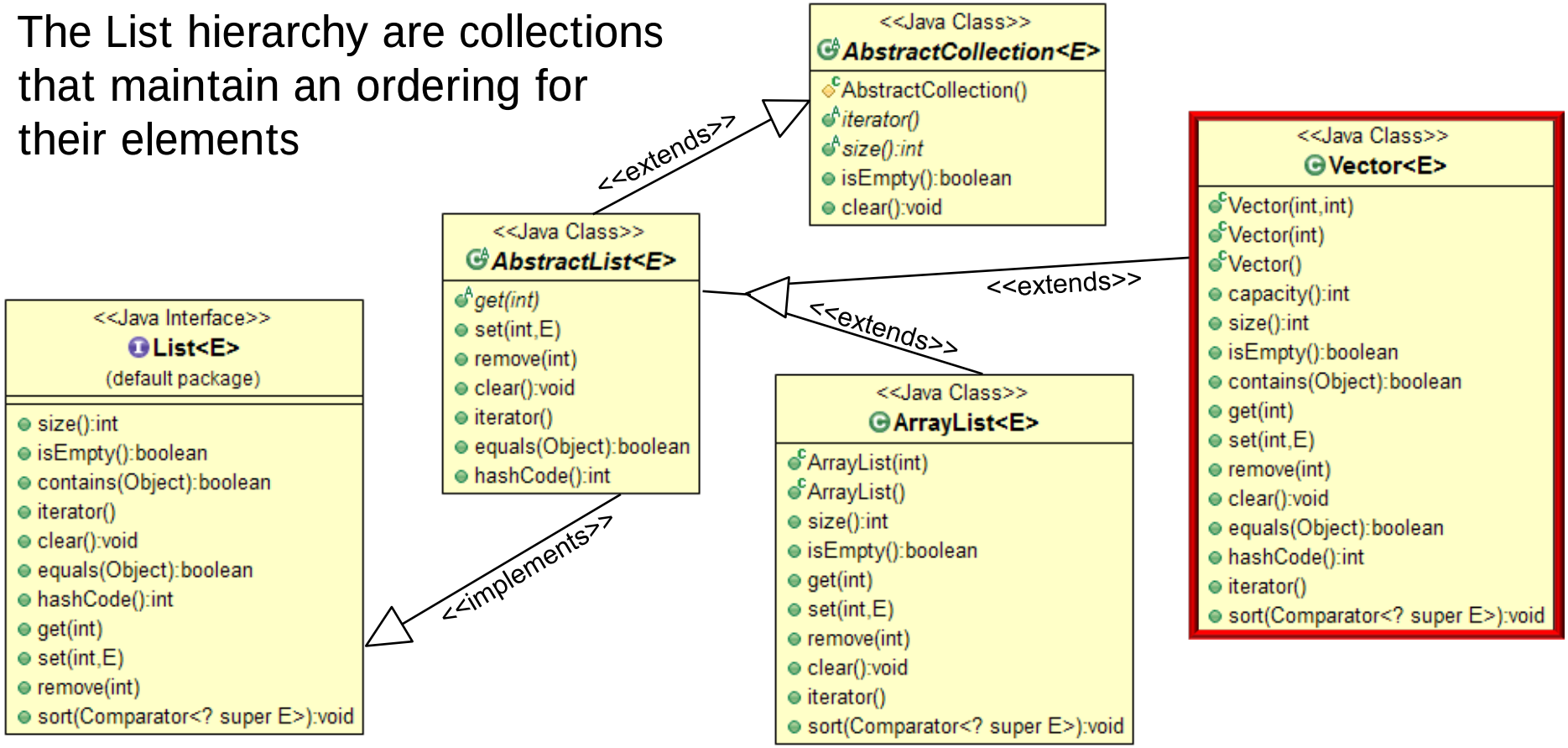
- The List hierarchy are collections that maintain an ordering for their elements



See developer.android.com/reference/java/util/AbstractList.html

Overview of Java's Support for Inheritance

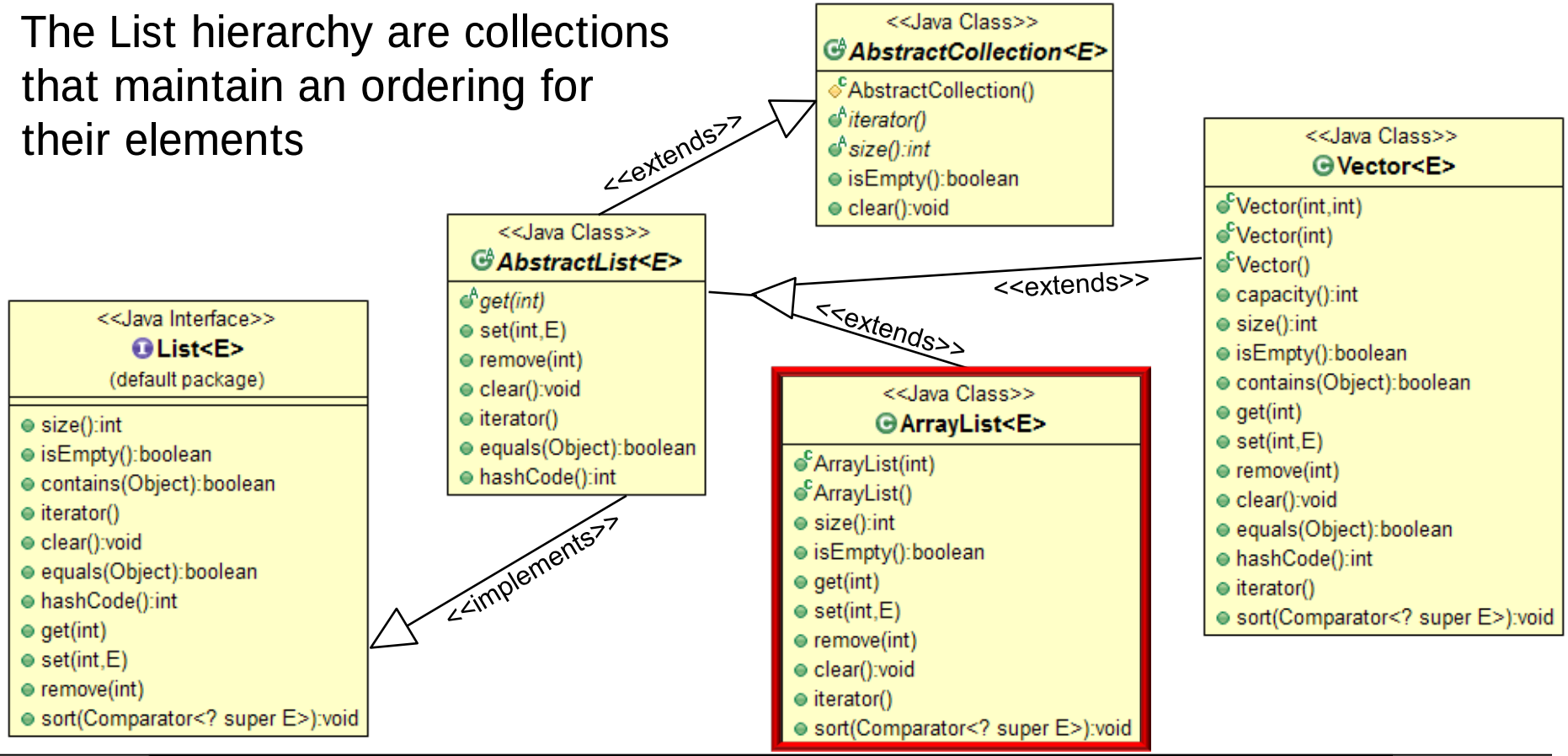
- The List hierarchy are collections that maintain an ordering for their elements



See developer.android.com/reference/java/util/Vector.html

Overview of Java's Support for Inheritance

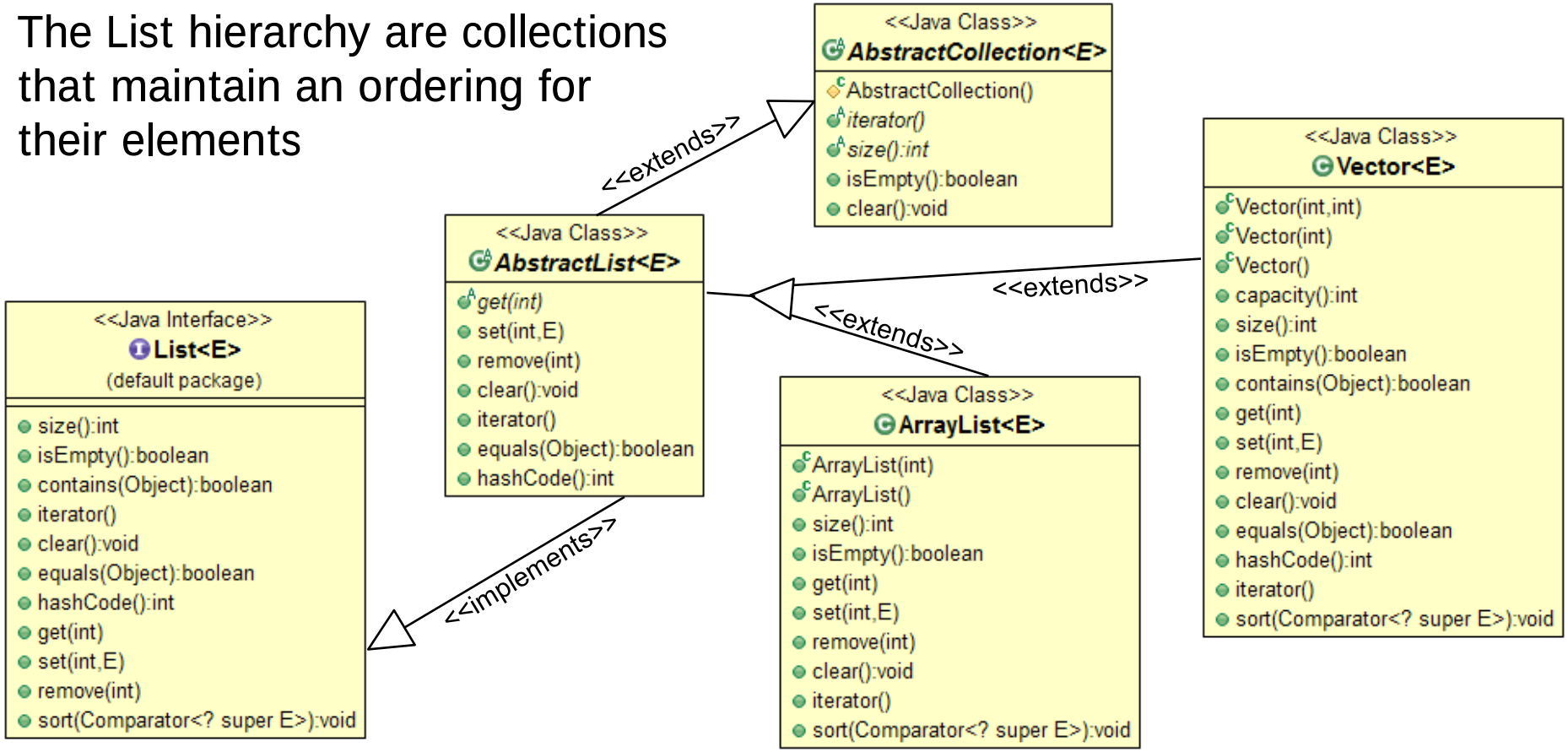
- The List hierarchy are collections that maintain an ordering for their elements



See developer.android.com/reference/java/util/ArrayList.html

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

Overview of Java's Support for Inheritance (Part 2)

Overview of Java's Support for Inheritance

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

```
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 developer.android.com/reference/java/lang/Object.html

Overview of Java's Support for Inheritance

- 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 developer.android.com/reference/java/lang/Object.html

Overview of Java's Support for Inheritance

- 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 developer.android.com/reference/java/lang/Object.html#hashCode()

Overview of Java's Support for Inheritance

- 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 developer.android.com/reference/java/lang/Object.html#equals(java.lang.Object)

Overview of Java's Support for Inheritance

- 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/tutorial/essential/concurrency/guardmeth.html

Overview of Java's Support for Inheritance

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

```
package java.lang;

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

Overview of Java's Support for Inheritance

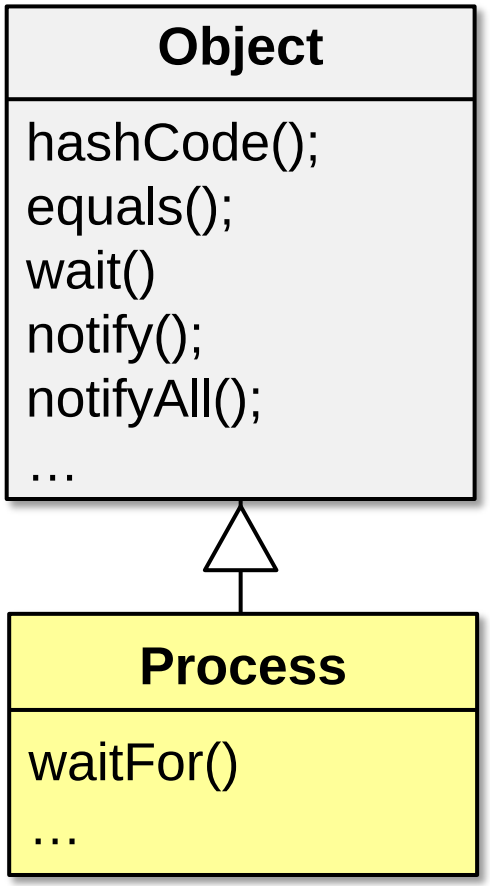
- Subclasses that don't explicitly extend a super class implicitly inherit from java.lang.Object, e.g.
  - java.lang.Process implicitly extends java.lang.Object

```
package java.lang;

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

See developer.android.com/reference/java/lang/Process.html

Overview of Java's Support for Inheritance

- 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

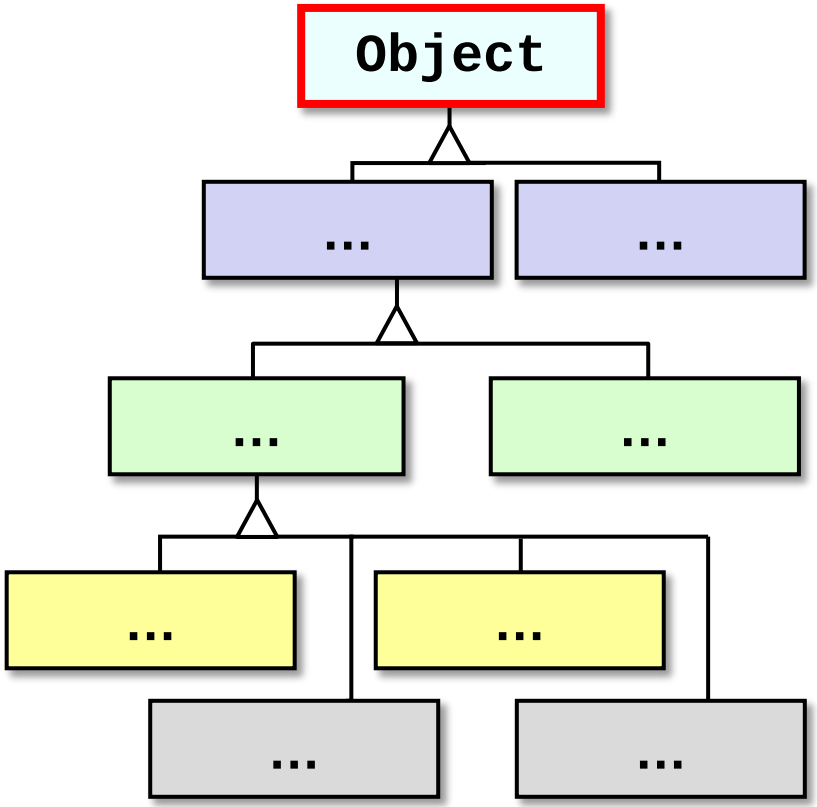


Overview of Java's Support for Inheritance

- java.lang.Object is the most general of all classes

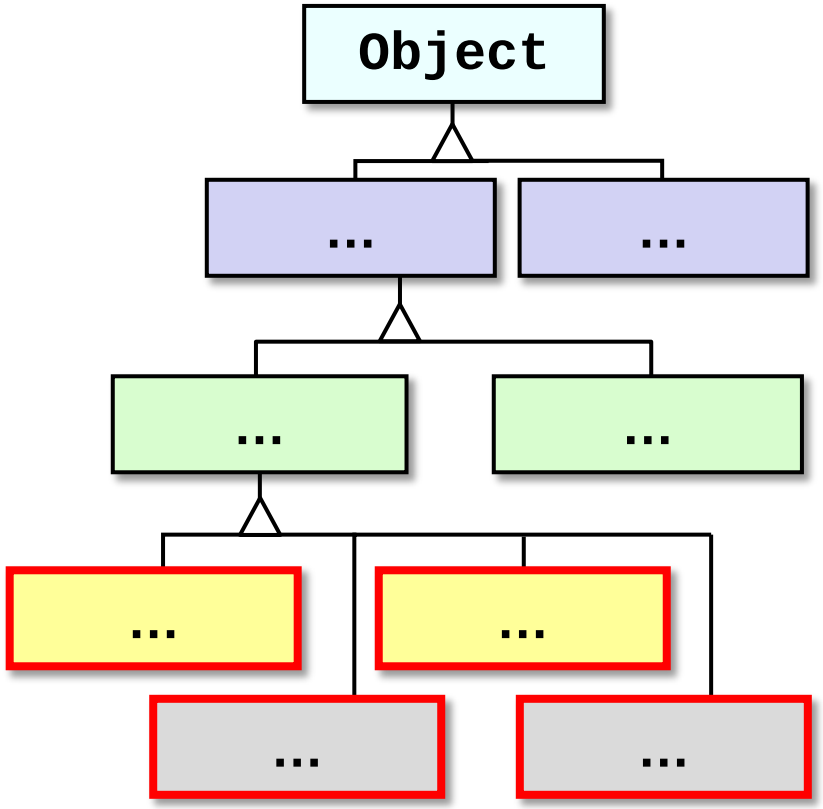
Overview of Java's Support for Inheritance

- java.lang.Object is the most general of all classes
  - It serves as the root of a hierarchy of classes available to Java apps



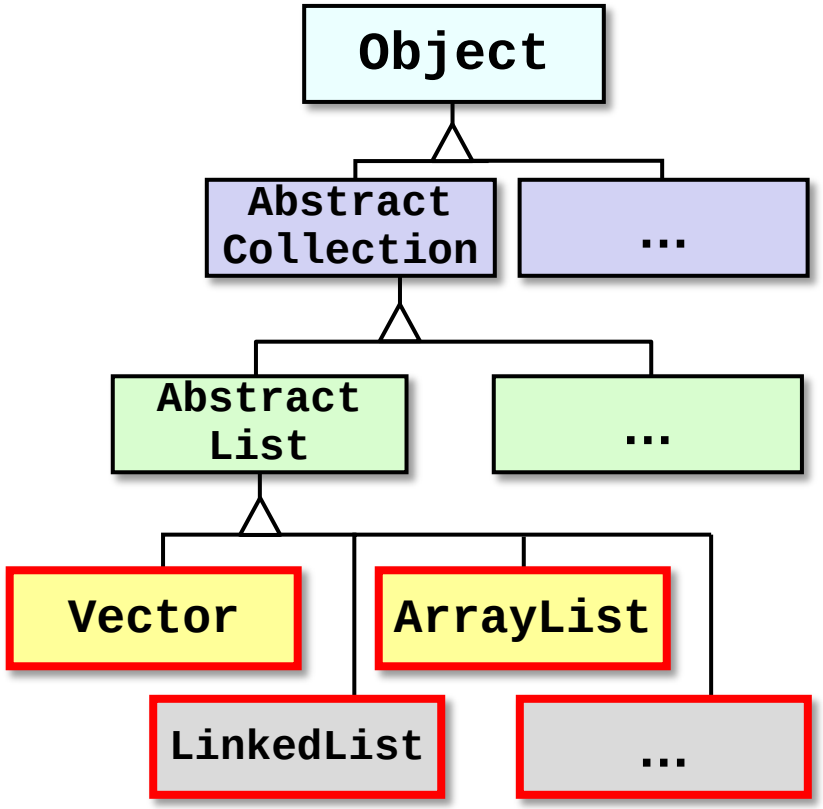
Overview of Java's Support for Inheritance

- 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



Overview of Java's Support for Inheritance

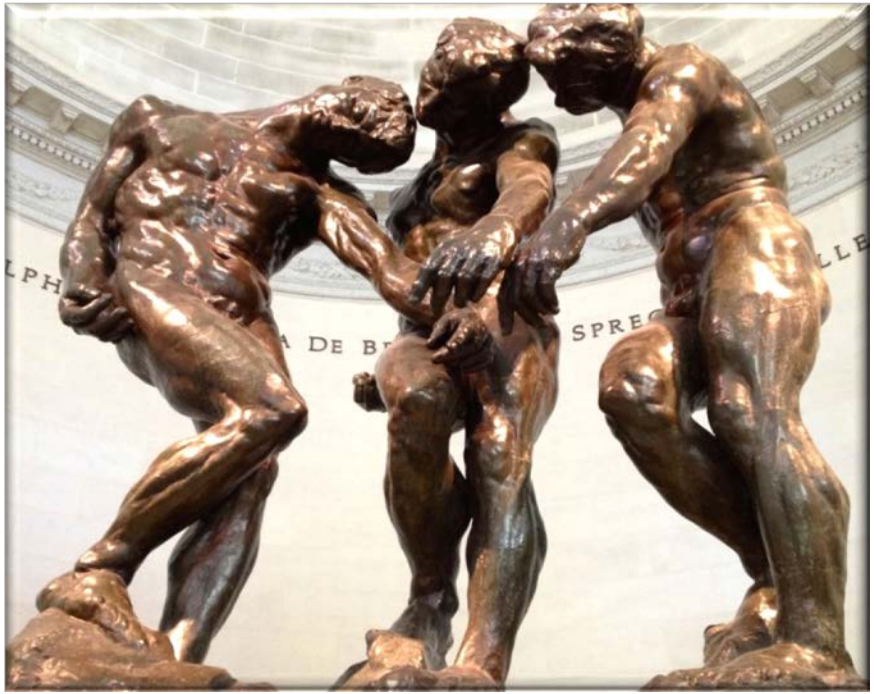
- java.lang.Object is the most general of all classes
  - It serves as the root of a hierarchy of classes available to Java apps
  - Classes towards the bottom of the inheritance hierarchy are more specialized
    - e.g., List-related subclasses override methods inherited from super classes



Overview of Java's Support for Inheritance

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

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

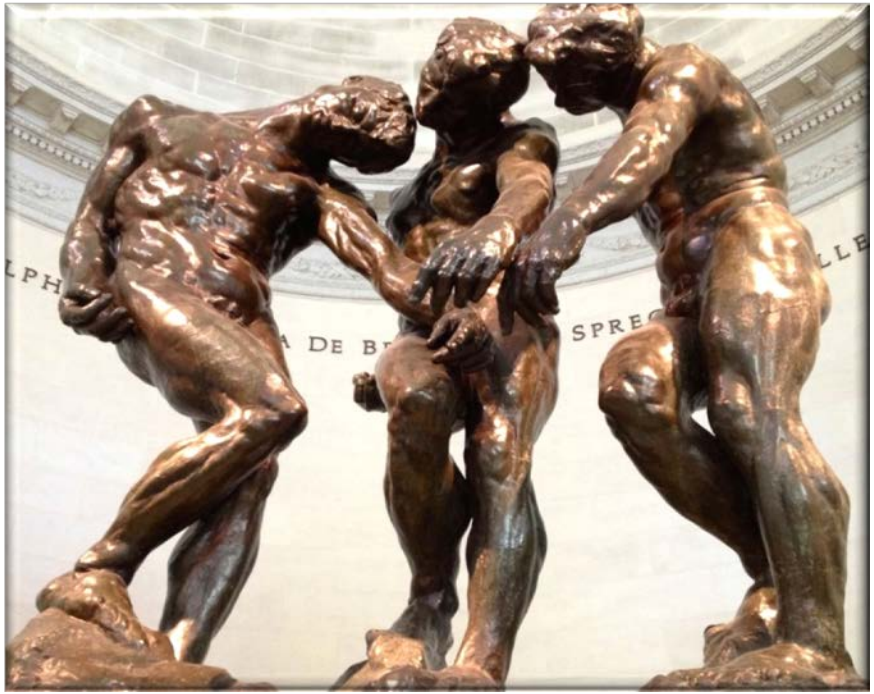


Overview of Java's Support for Inheritance

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

```
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



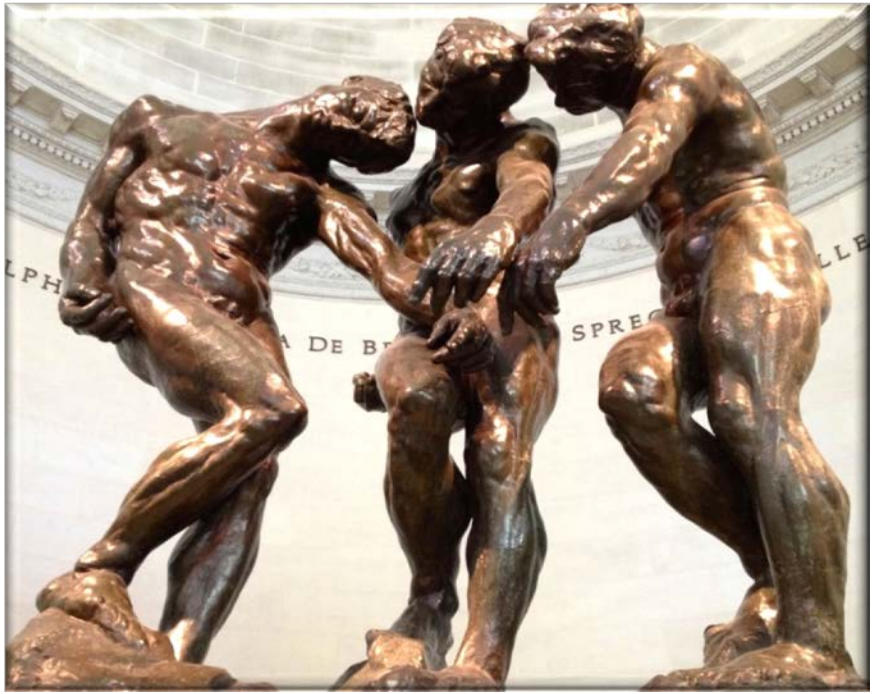
See developer.android.com/reference/java/util/Stack.html

Overview of Java's Support for Inheritance

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

```
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 developer.android.com/reference/java/util/Vector.html

Overview of Java's Support for Inheritance

- 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> {
```

Overview of Java's Support for Inheritance

- 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> {
```

Overview of Java's Support for Inheritance

- 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();
```

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

Overview of Java's Support for Inheritance

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

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

Overview of Java's Support for Inheritance

- Subclass methods inherited from a super class are used for 3 purposes
  1. To augment the subclass API
  2. To implement subclass methods
     - e.g., Stack's push() method is implemented via addElement() method inherited from Vector

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

Overview of Java's Support for Inheritance

- Subclass methods inherited from a super class are used for 3 purposes
  1. To augment the subclass API
  2. To implement subclass methods
     - e.g., Stack's push() method is implemented via addElement() method inherited from Vector

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

Overview of Java's Support for Inheritance

- 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

```
public abstract class
        AbstractMap<K,V> ...
  public abstract
    Set<Entry<K,V>> entrySet();
  public V put(K key, V value)
  { ... }
  ...

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

See en.wikipedia.org/wiki/Type_signature#Java_2

Overview of Java's Support for Inheritance

- 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
     - e.g., the HashMap subclass overrides AbstractMap super class methods

```
public abstract class
        AbstractMap<K,V> ...
  public abstract
    Set<Entry<K,V>> entrySet();
  public V put(K key, V value)
  { ... }
  ...

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

See docs.oracle.com/javase/tutorial/java/IandI/override.html

Overview of Java's Support for Inheritance

- 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
     - e.g., the HashMap subclass overrides AbstractMap super class methods

```
public abstract class
        AbstractMap<K,V> ...
  public abstract
    Set<Entry<K,V>> entrySet();
  public V put(K key, V value)
  { ... }
  ...

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

See docs.oracle.com/javase/tutorial/java/IandI/override.html

Overview of Java's Support for Inheritance

- 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
     - e.g., the HashMap subclass overrides AbstractMap super class methods

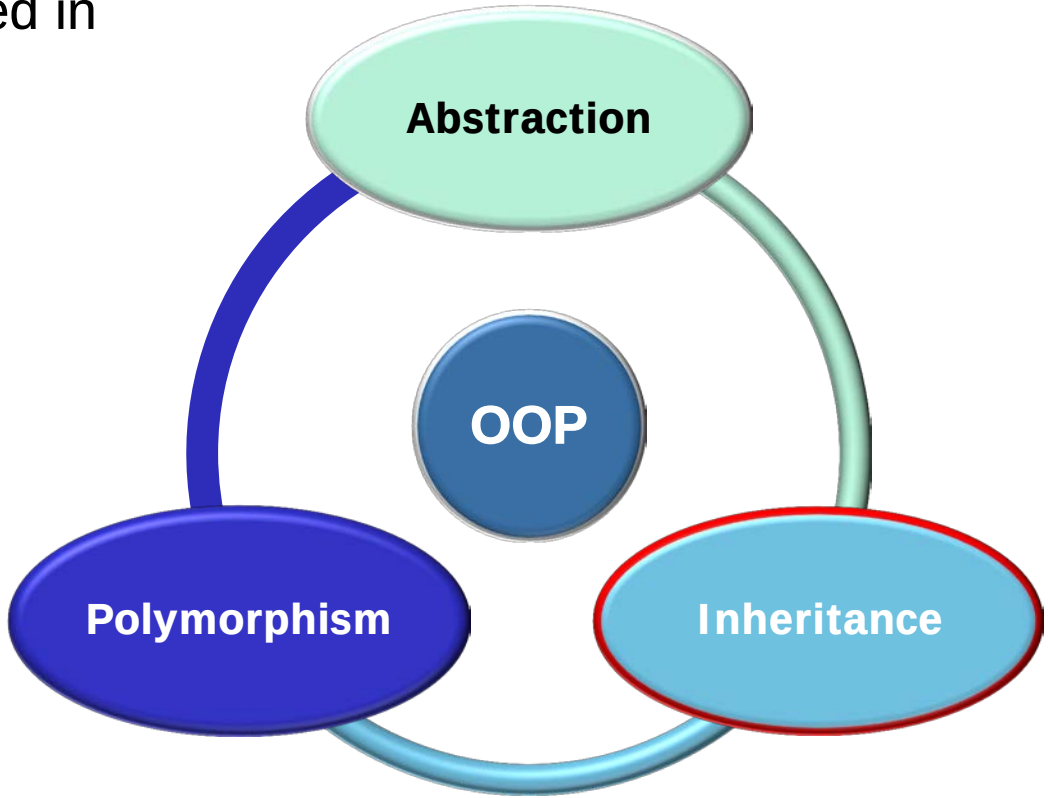
```
public abstract class
        AbstractMap<K,V> ...
  public abstract
    Set<Entry<K,V>> entrySet();
  public V put(K key, V value)
  { ... }
  ...

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

Method overriding is covered next in our discussion of Java Polymorphism

Overview of Java's Support for Polymorphism (Part 1)

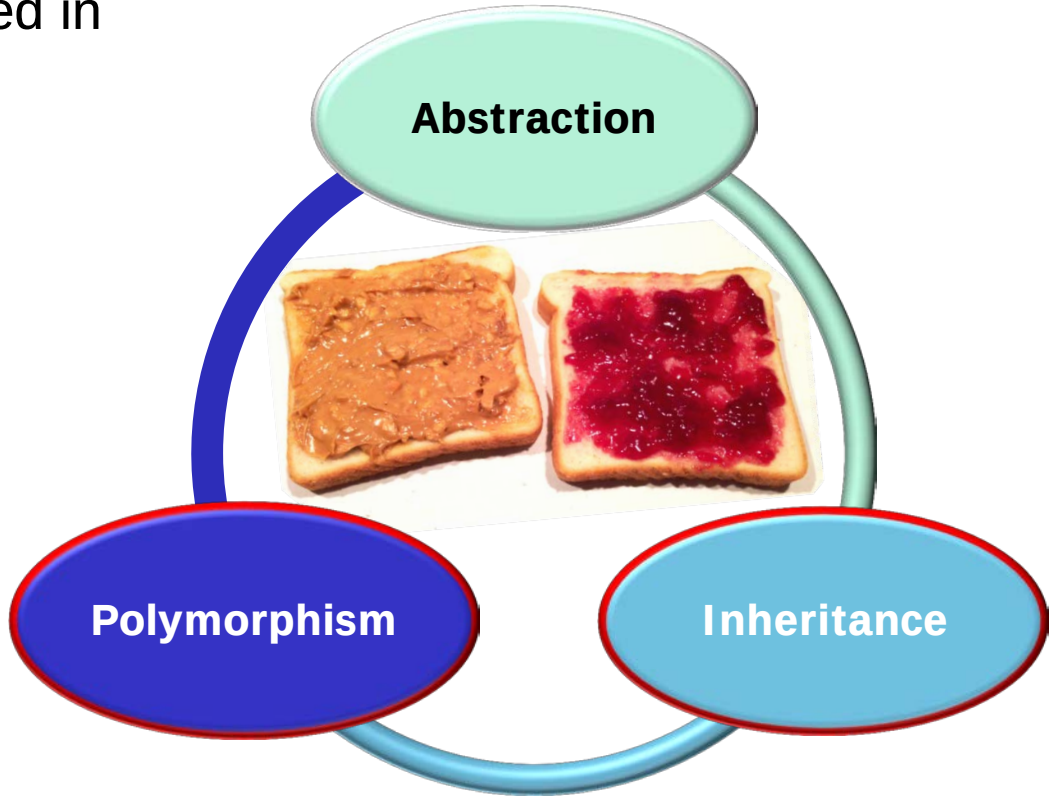
Overview of Java's Support for Polymorphism

- Inheritance is nearly always used in conjunction with polymorphism



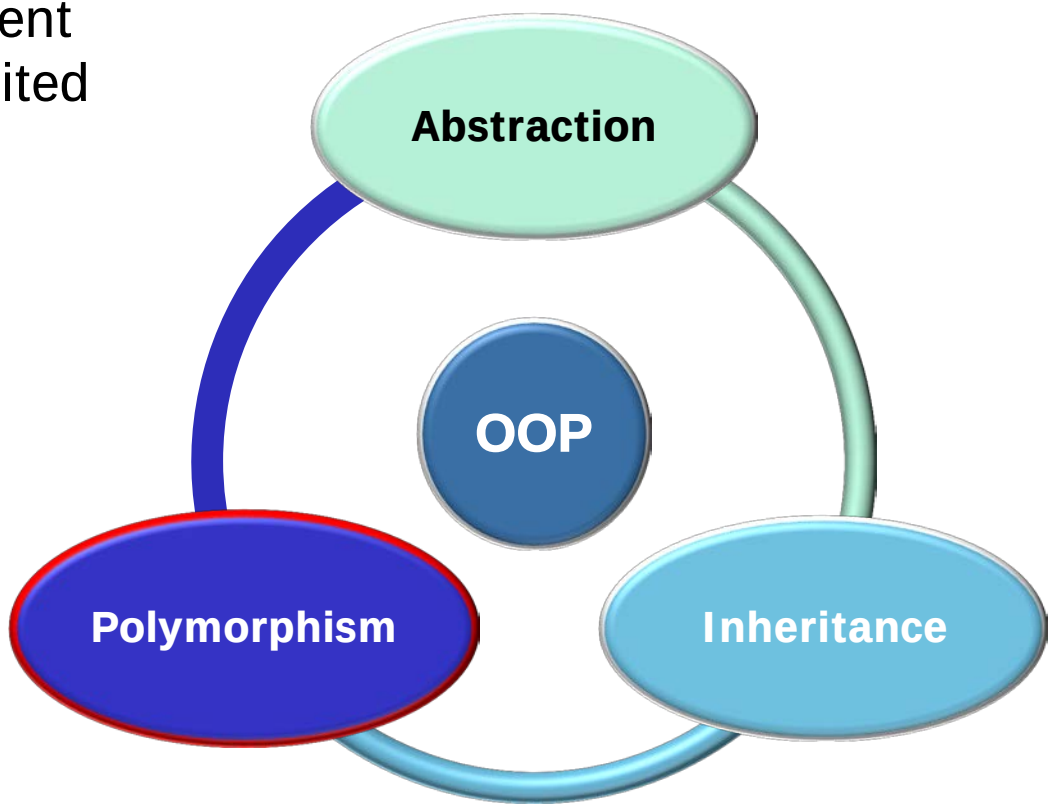
Overview of Java's Support for Polymorphism

- Inheritance is nearly always used in conjunction with polymorphism



Overview of Java's Support for Polymorphism

- Polymorphism enables transparent customization of methods inherited from a super class



See en.wikipedia.org/wiki/Polymorphism_(computer_science)

Overview of Java's Support for Polymorphism

- Polymorphism & inheritance are essential to the "open/closed principle"

See en.wikipedia.org/wiki/Open/closed_principle

Overview of Java's Support for Polymorphism

- Polymorphism & inheritance are essential to the "open/closed principle"
  - "A class should be open for extension, but closed for modification"

See en.wikipedia.org/wiki/Open/closed_principle

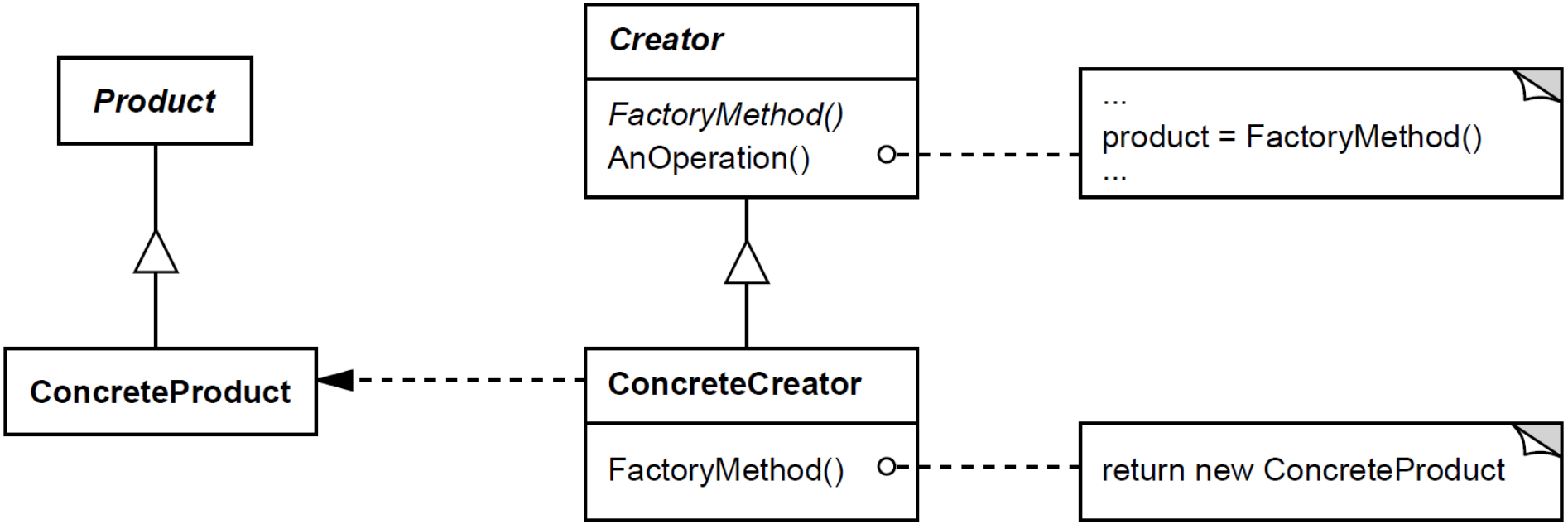
Overview of Java's Support for Polymorphism

- Polymorphism & inheritance are essential to the "open/closed principle"
  - "A class should be open for extension, but closed for modification"
  - Insulating a class from modifications helps make the class more robust, flexible, & reusable

See www.dre.vanderbilt.edu/~schmidt/OCP.pdf

Overview of Java's Support for Polymorphism

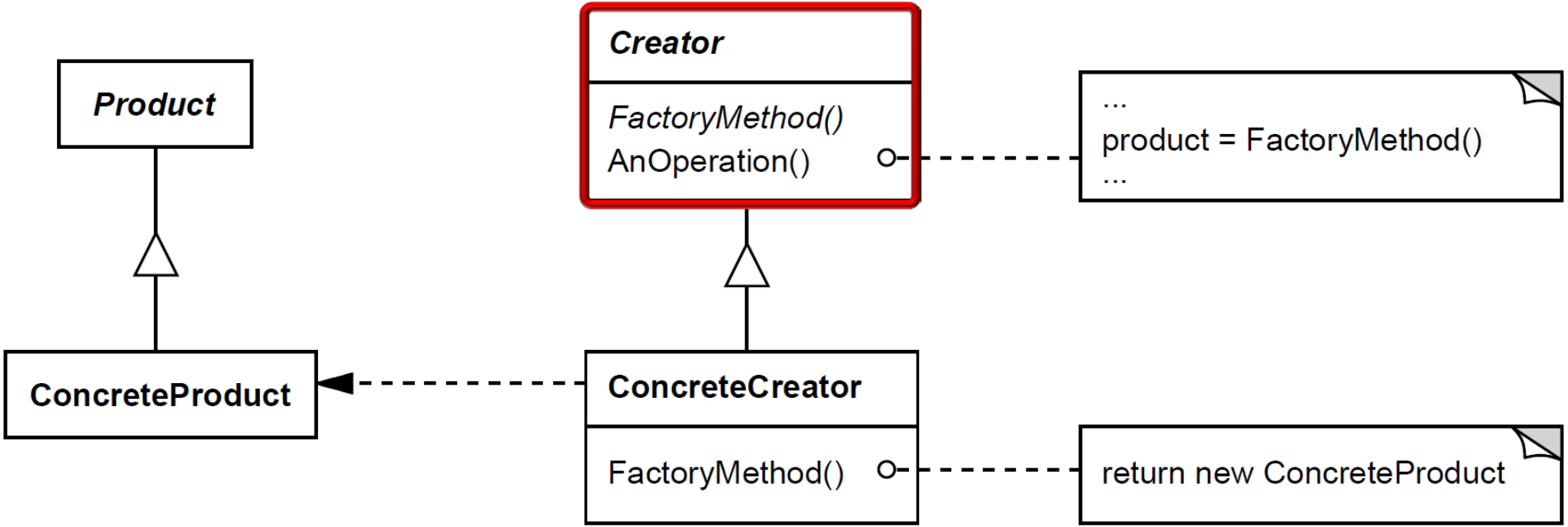
- The "open/closed principle" can be applied in conjunction with patterns to enable extensions *without* modifying existing classes or apps



See en.wikipedia.org/wiki/Factory_method_pattern

Overview of Java's Support for Polymorphism

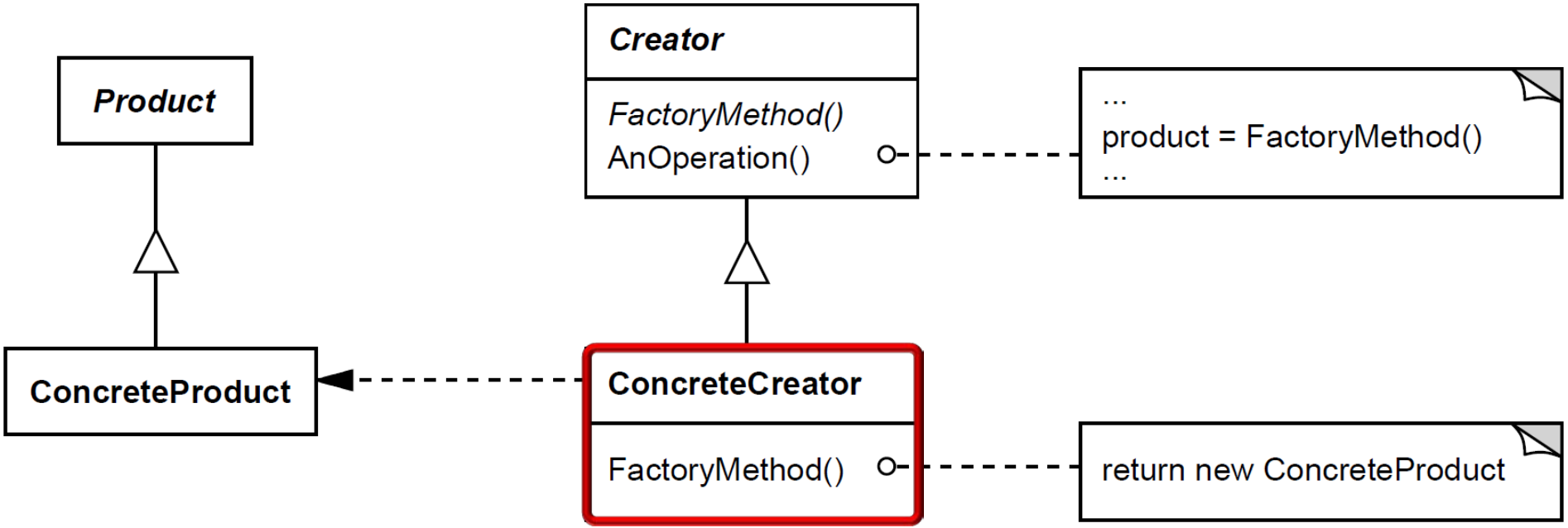
- The "open/closed principle" can be applied in conjunction with patterns to enable extensions *without* modifying existing classes or apps



See en.wikipedia.org/wiki/Factory_method_pattern

Overview of Java's Support for Polymorphism

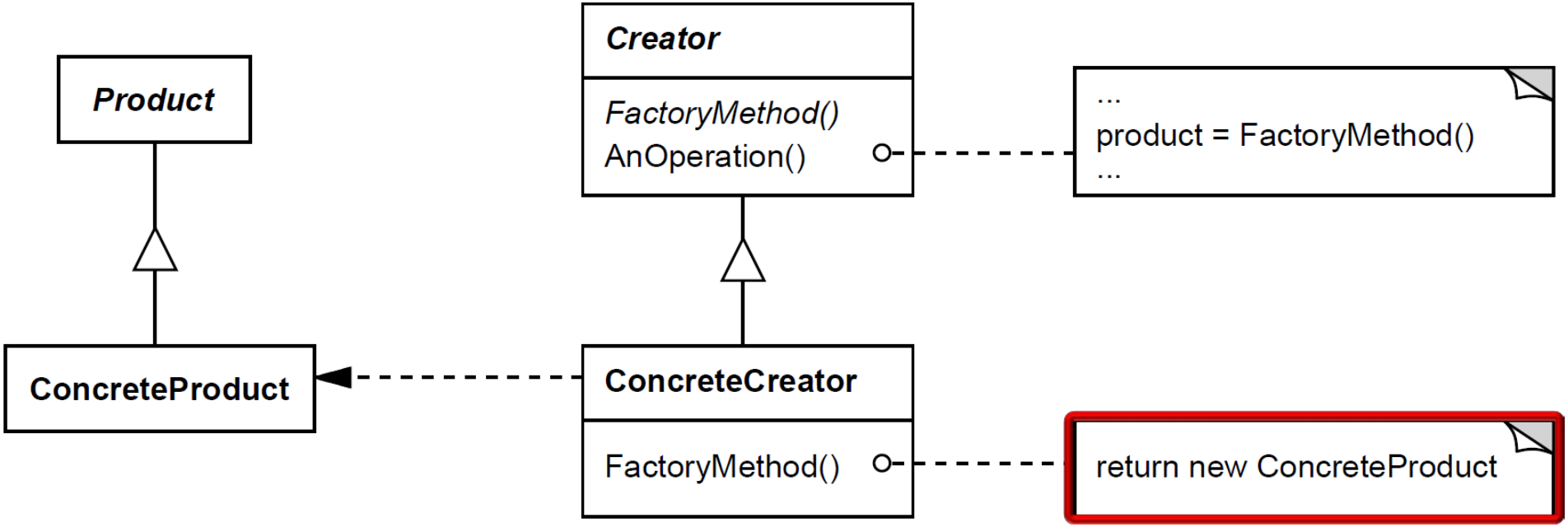
- The "open/closed principle" can be applied in conjunction with patterns to enable extensions *without* modifying existing classes or apps



See en.wikipedia.org/wiki/Factory_method_pattern

Overview of Java's Support for Polymorphism

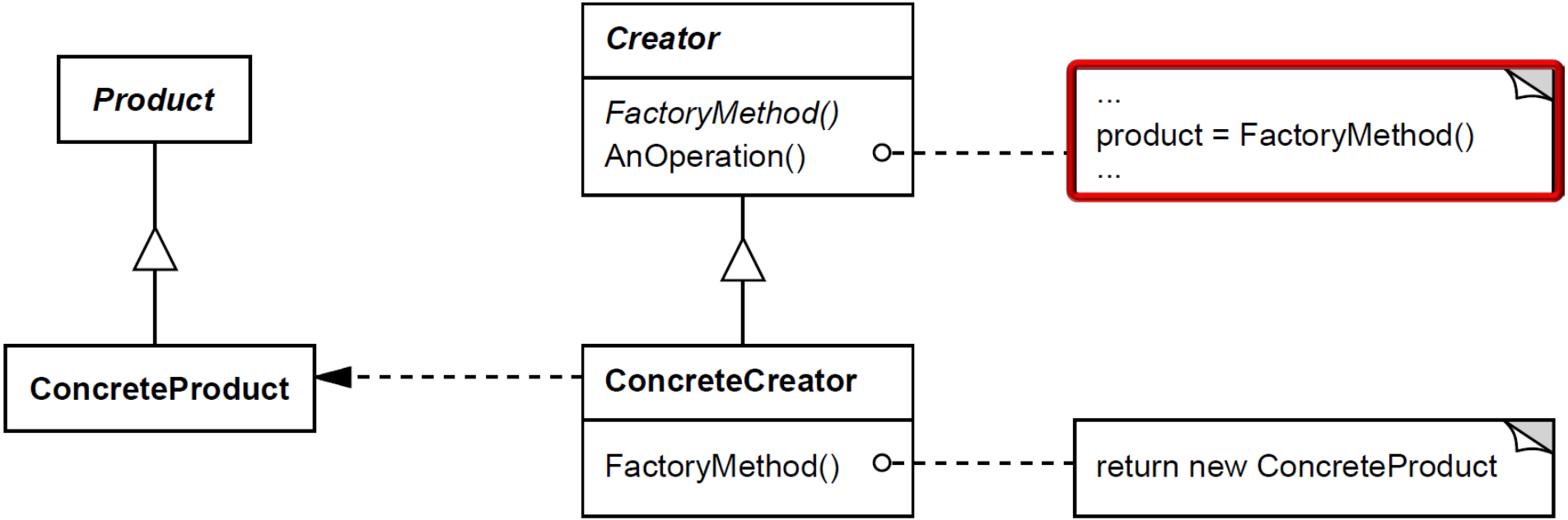
- The "open/closed principle" can be applied in conjunction with patterns to enable extensions *without* modifying existing classes or apps



See en.wikipedia.org/wiki/Factory_method_pattern

Overview of Java's Support for Polymorphism

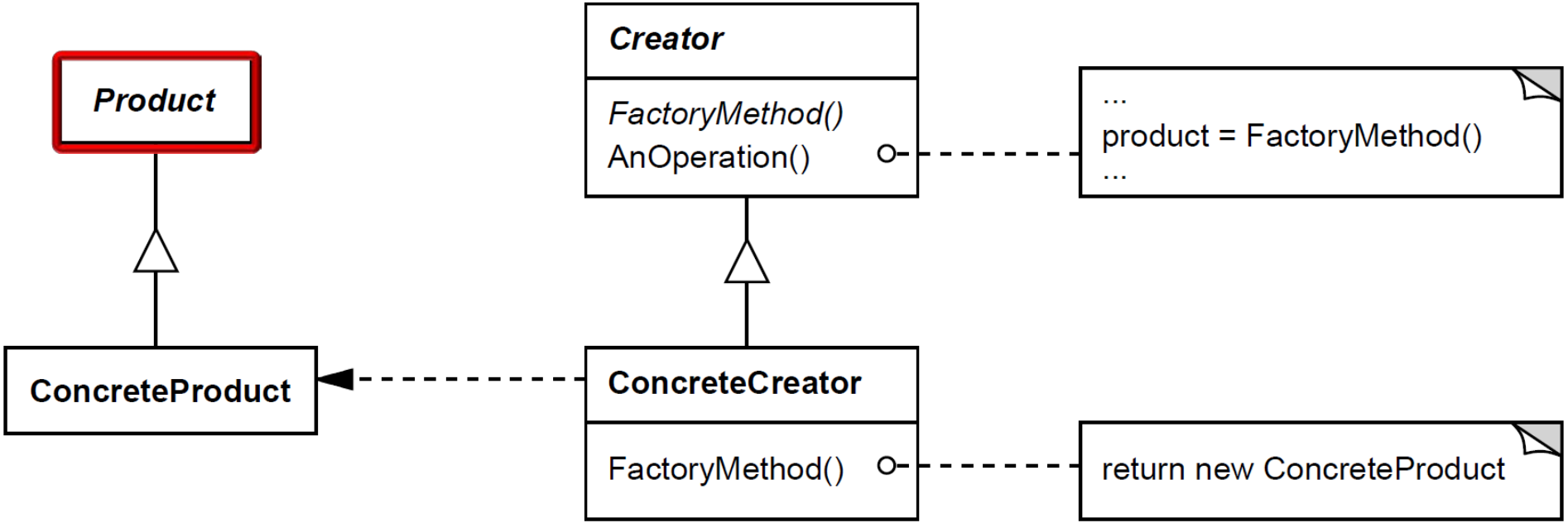
- The "open/closed principle" can be applied in conjunction with patterns to enable extensions *without* modifying existing classes or apps



See en.wikipedia.org/wiki/Factory_method_pattern

Overview of Java's Support for Polymorphism

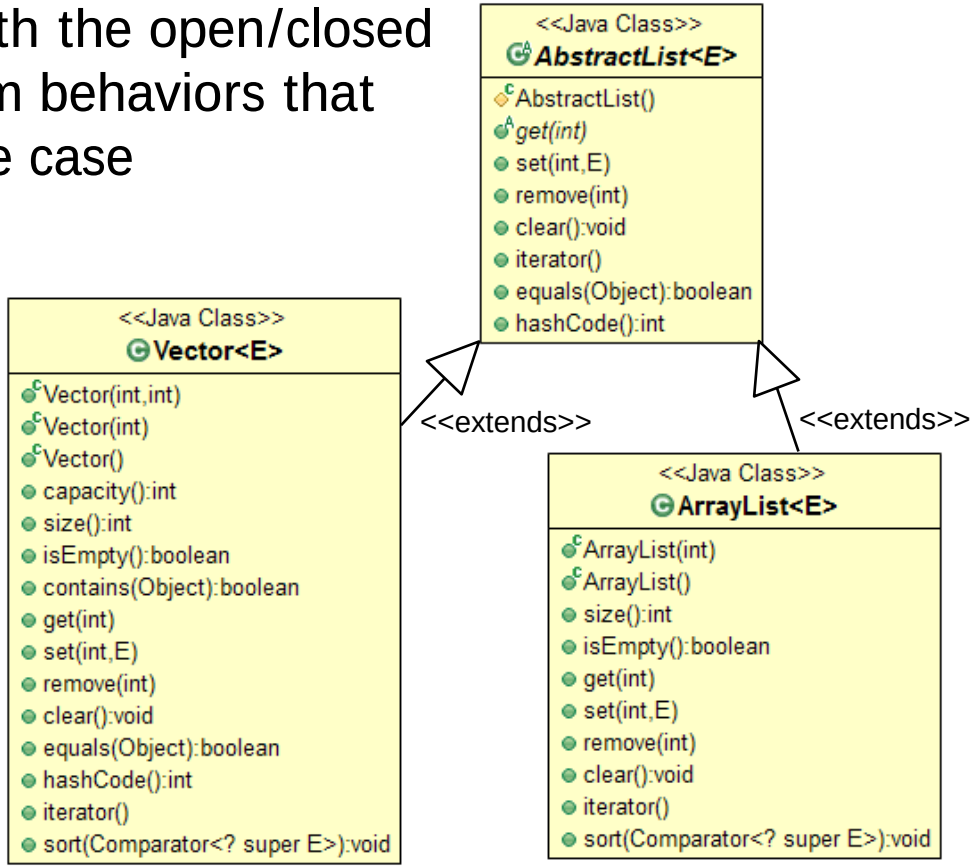
- The "open/closed principle" can be applied in conjunction with patterns to enable extensions *without* modifying existing classes or apps



See en.wikipedia.org/wiki/Factory_method_pattern

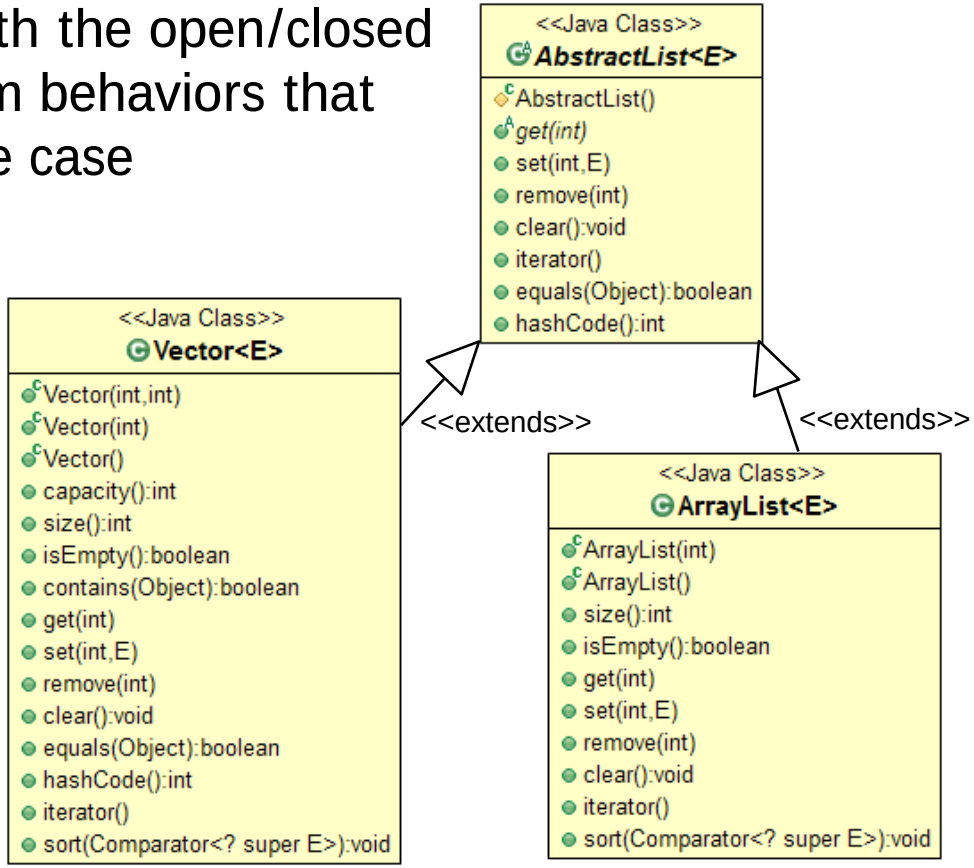
Overview of Java's Support for Polymorphism

- Subclasses defined in accordance with the open/closed principle can define their own custom behaviors that are more suitable for a particular use case



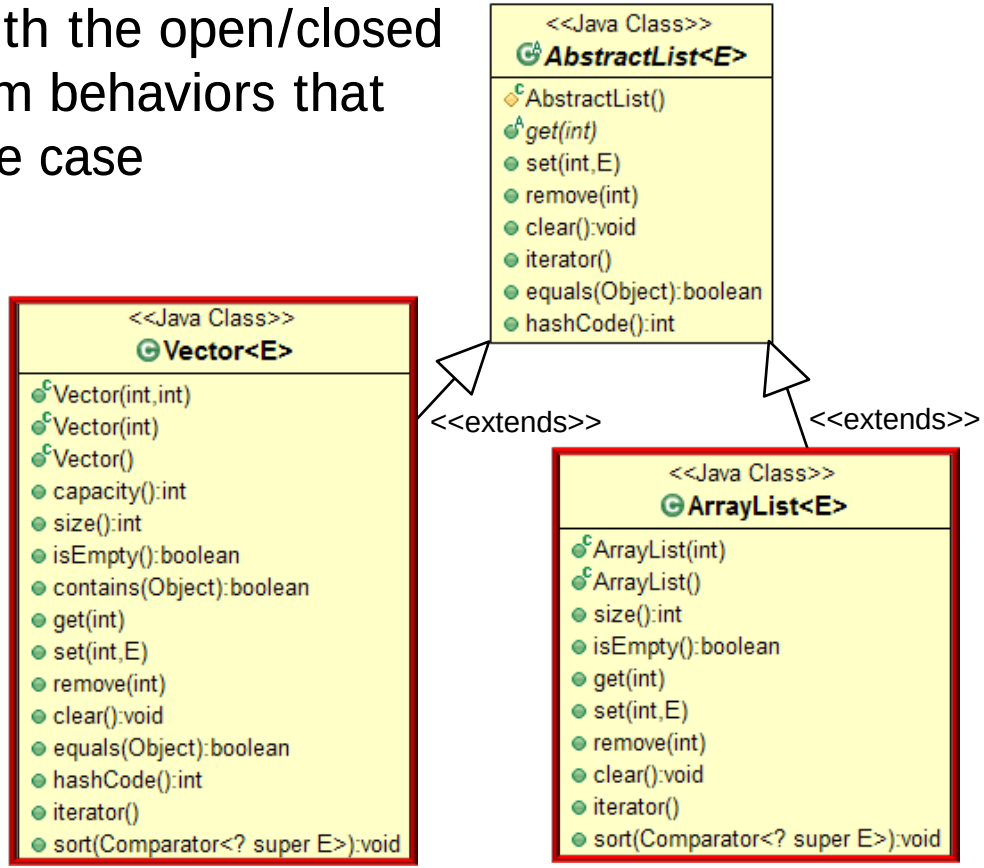
Overview of Java's Support for Polymorphism

- Subclasses defined in accordance with the open/closed principle can define their own custom behaviors that are more suitable for a particular use case
  - While still reusing structure & functionality from super class



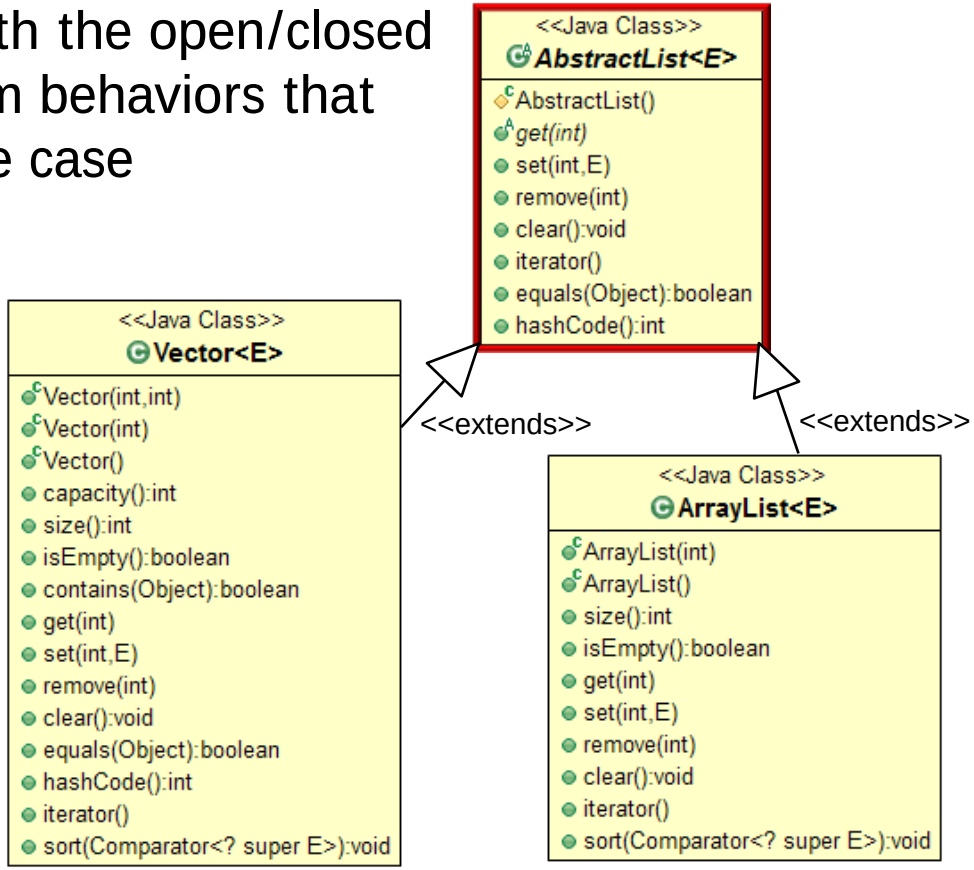
Overview of Java's Support for Polymorphism

- Subclasses defined in accordance with the open/closed principle can define their own custom behaviors that are more suitable for a particular use case
  - While still reusing structure & functionality from super class
    - e.g., Vector & ArrayList both inherit AbstractList methods



Overview of Java's Support for Polymorphism

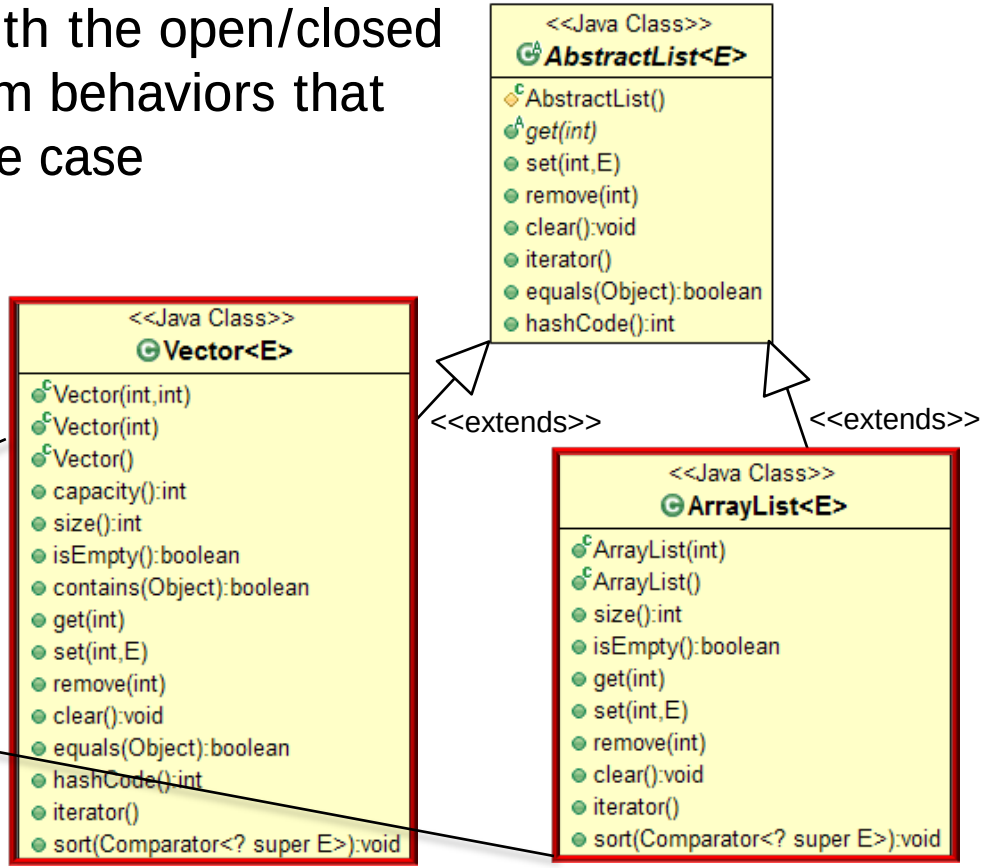
- Subclasses defined in accordance with the open/closed principle can define their own custom behaviors that are more suitable for a particular use case
  - While still reusing structure & functionality from super class
    - e.g., Vector & ArrayList both inherit AbstractList methods



Overview of Java's Support for Polymorphism (Part 2)

Overview of Java's Support for Polymorphism

- Subclasses defined in accordance with the open/closed principle can define their own custom behaviors that are more suitable for a particular use case
  - While still reusing structure & functionality from super class
    - e.g., Vector & ArrayList both inherit AbstractList methods

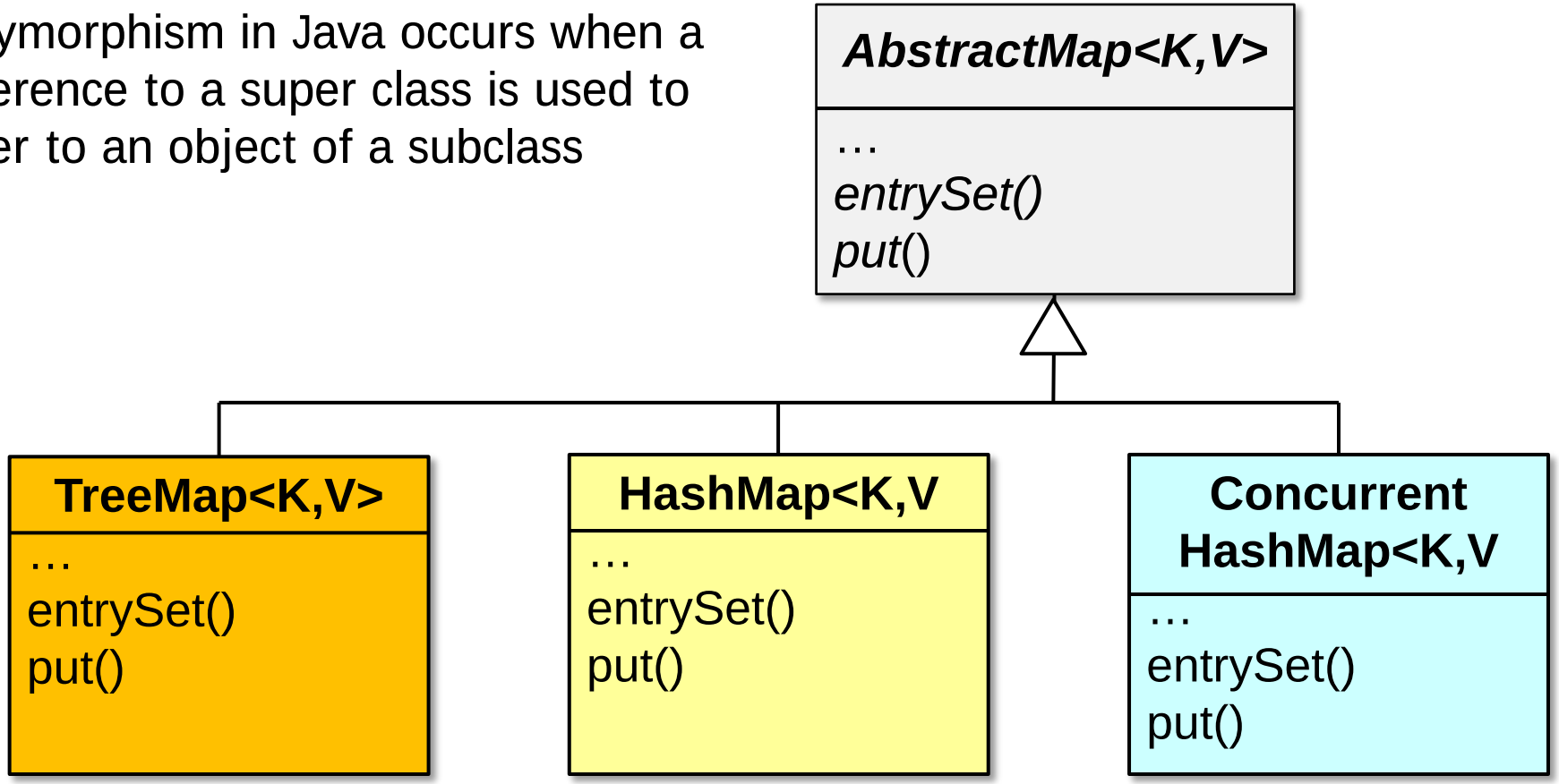


Methods in each subclass emphasize different List properties

See beginnersbook.com/2013/12/difference-between-arraylist-and-vector-in-java

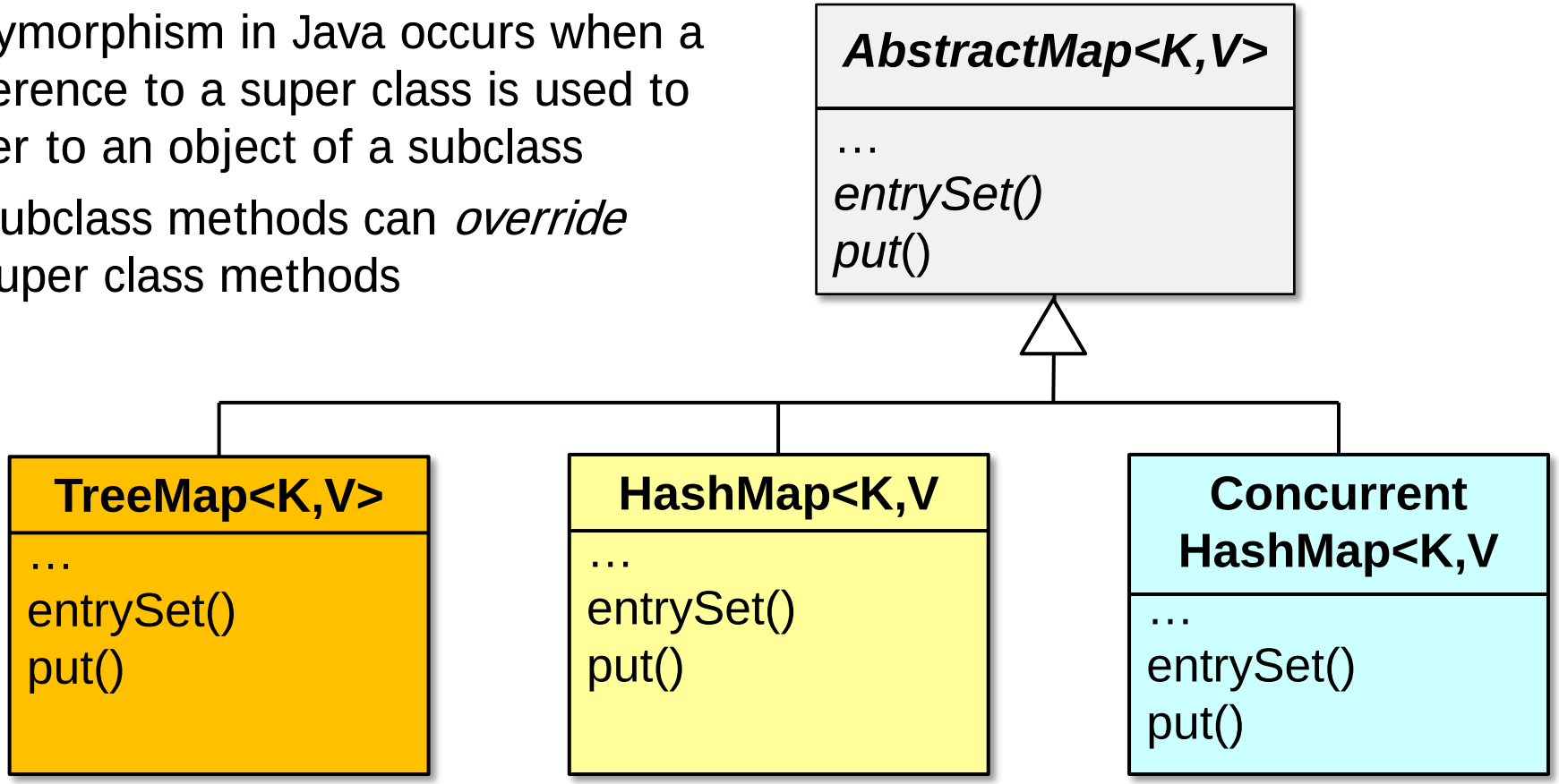
Overview of Java's Support for Polymorphism

- Polymorphism in Java occurs when a reference to a super class is used to refer to an object of a subclass



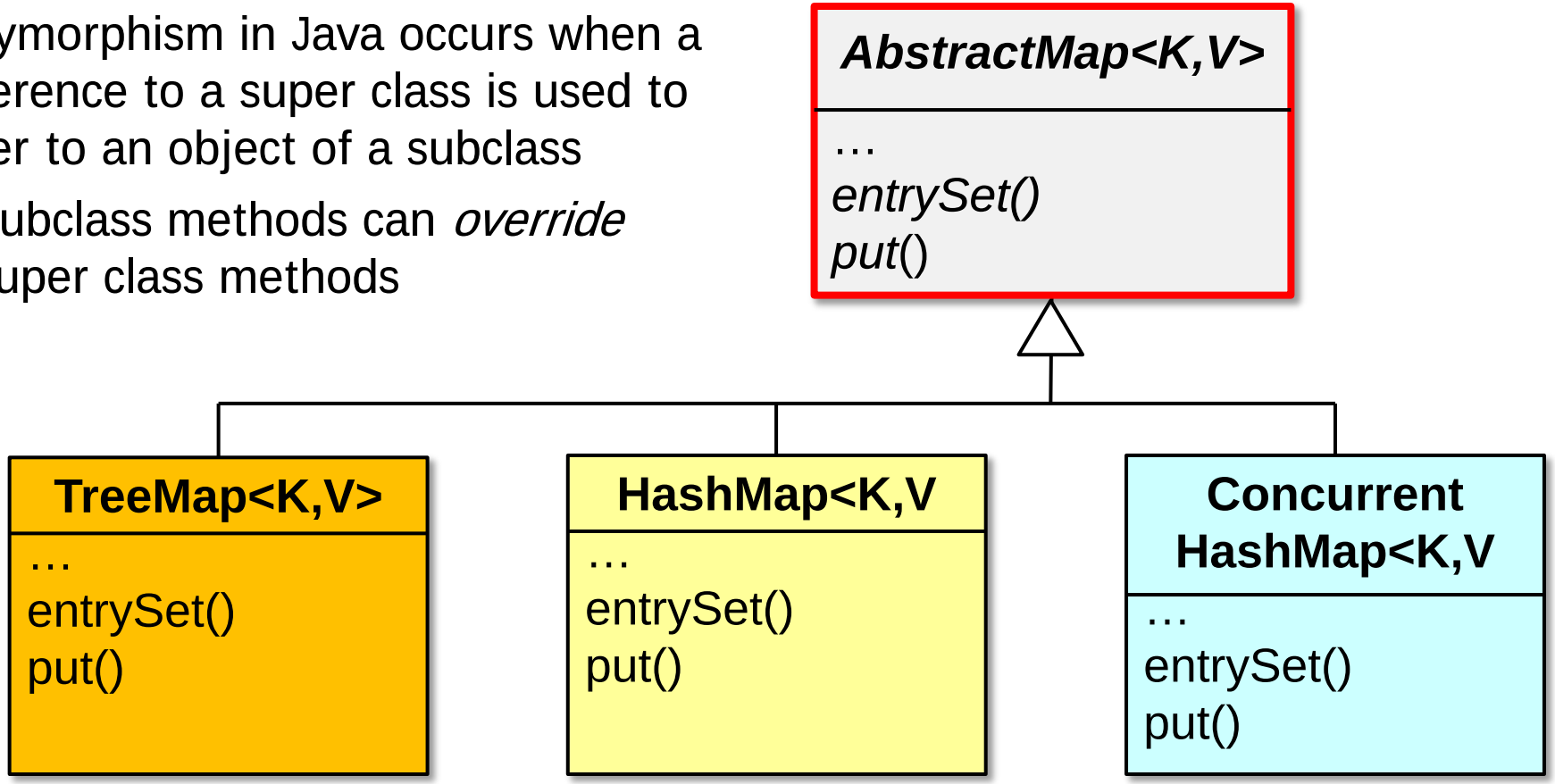
Overview of Java's Support for Polymorphism

- Polymorphism in Java occurs when a reference to a super class is used to refer to an object of a subclass
  - Subclass methods can *override* super class methods



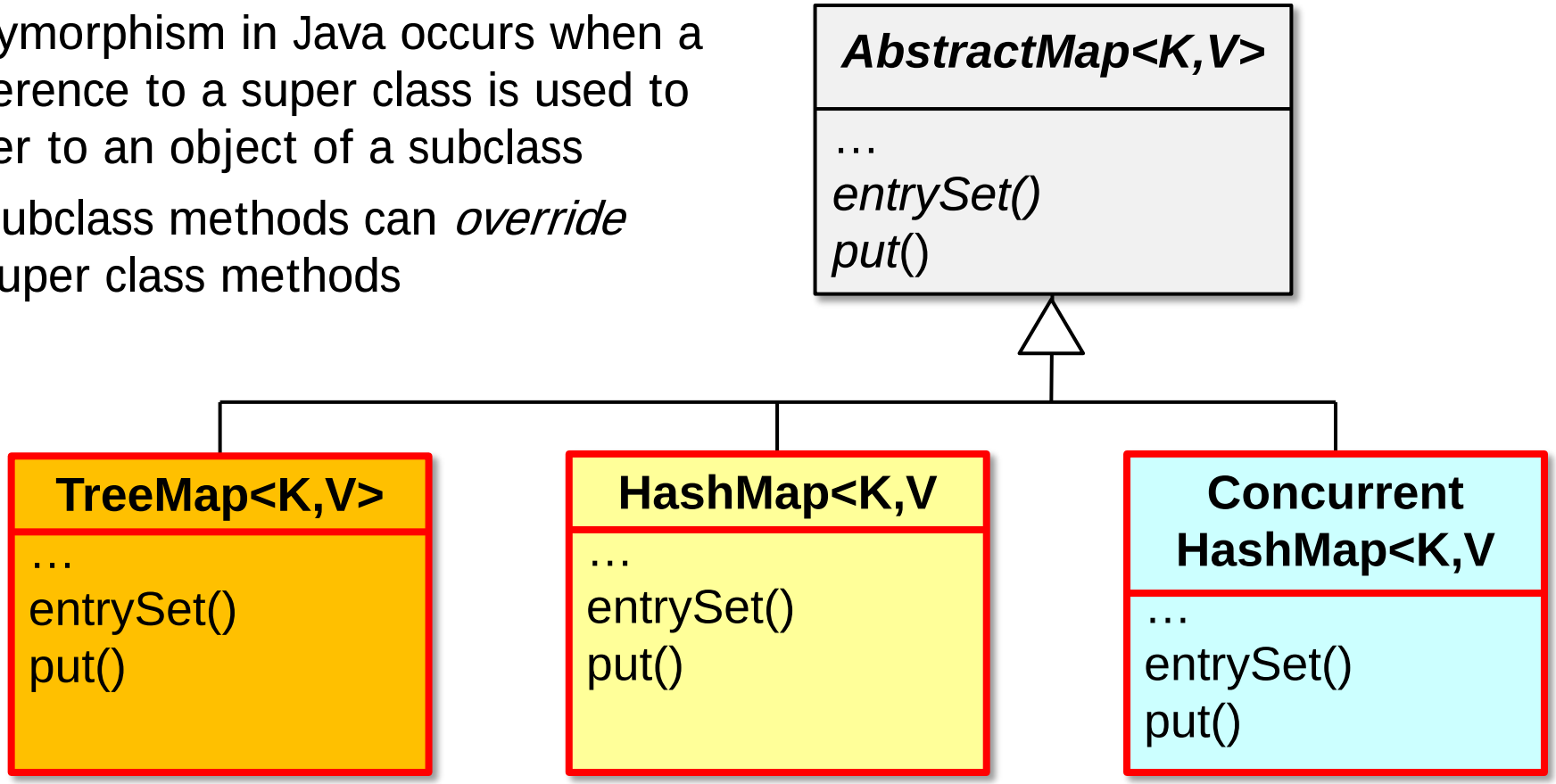
Overview of Java's Support for Polymorphism

- Polymorphism in Java occurs when a reference to a super class is used to refer to an object of a subclass
  - Subclass methods can *override* super class methods



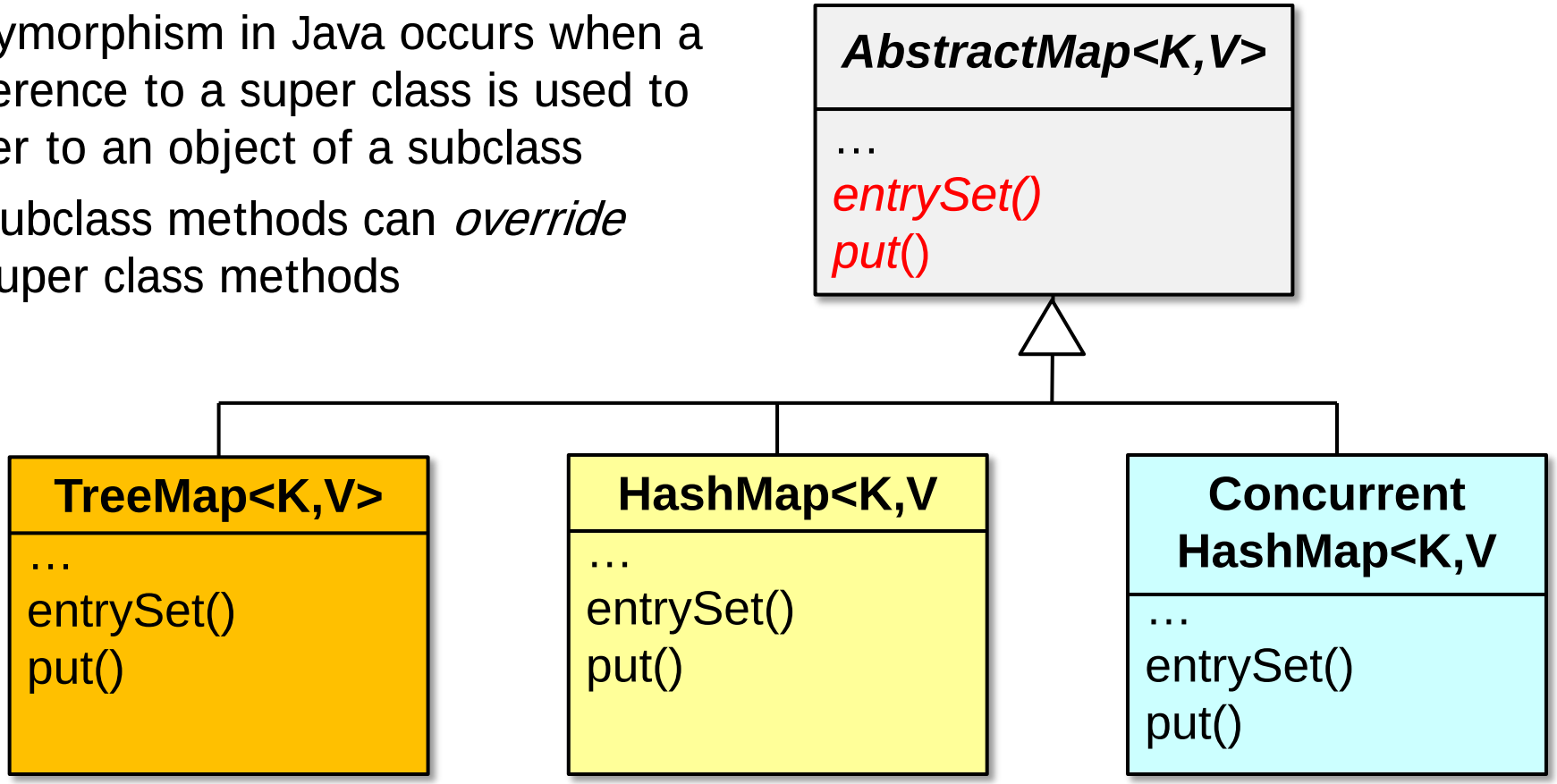
Overview of Java's Support for Polymorphism

- Polymorphism in Java occurs when a reference to a super class is used to refer to an object of a subclass
  - Subclass methods can *override* super class methods



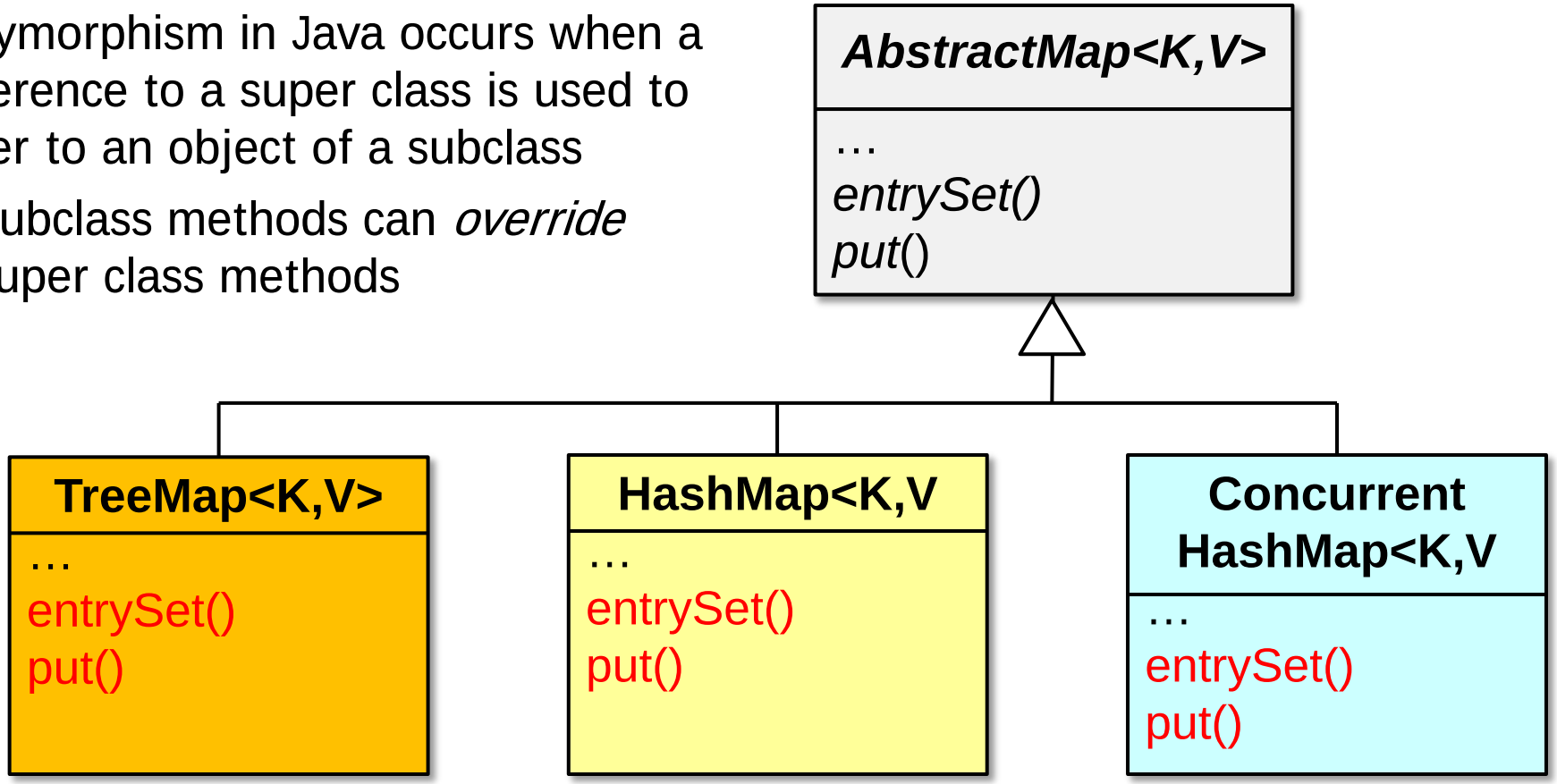
Overview of Java's Support for Polymorphism

- Polymorphism in Java occurs when a reference to a super class is used to refer to an object of a subclass
  - Subclass methods can *override* super class methods



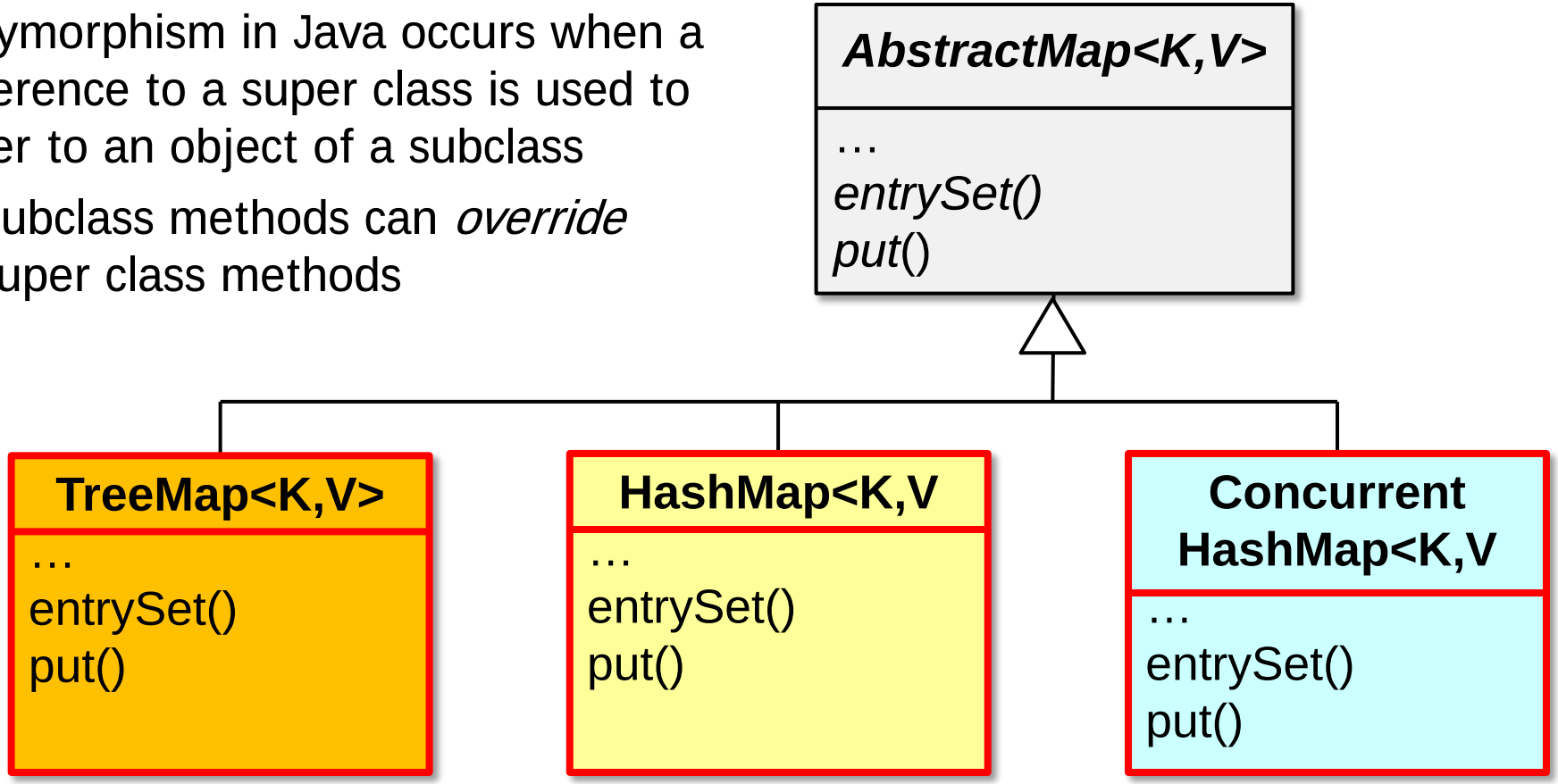
Overview of Java's Support for Polymorphism

- Polymorphism in Java occurs when a reference to a super class is used to refer to an object of a subclass
  - Subclass methods can *override* super class methods



Overview of Java's Support for Polymorphism

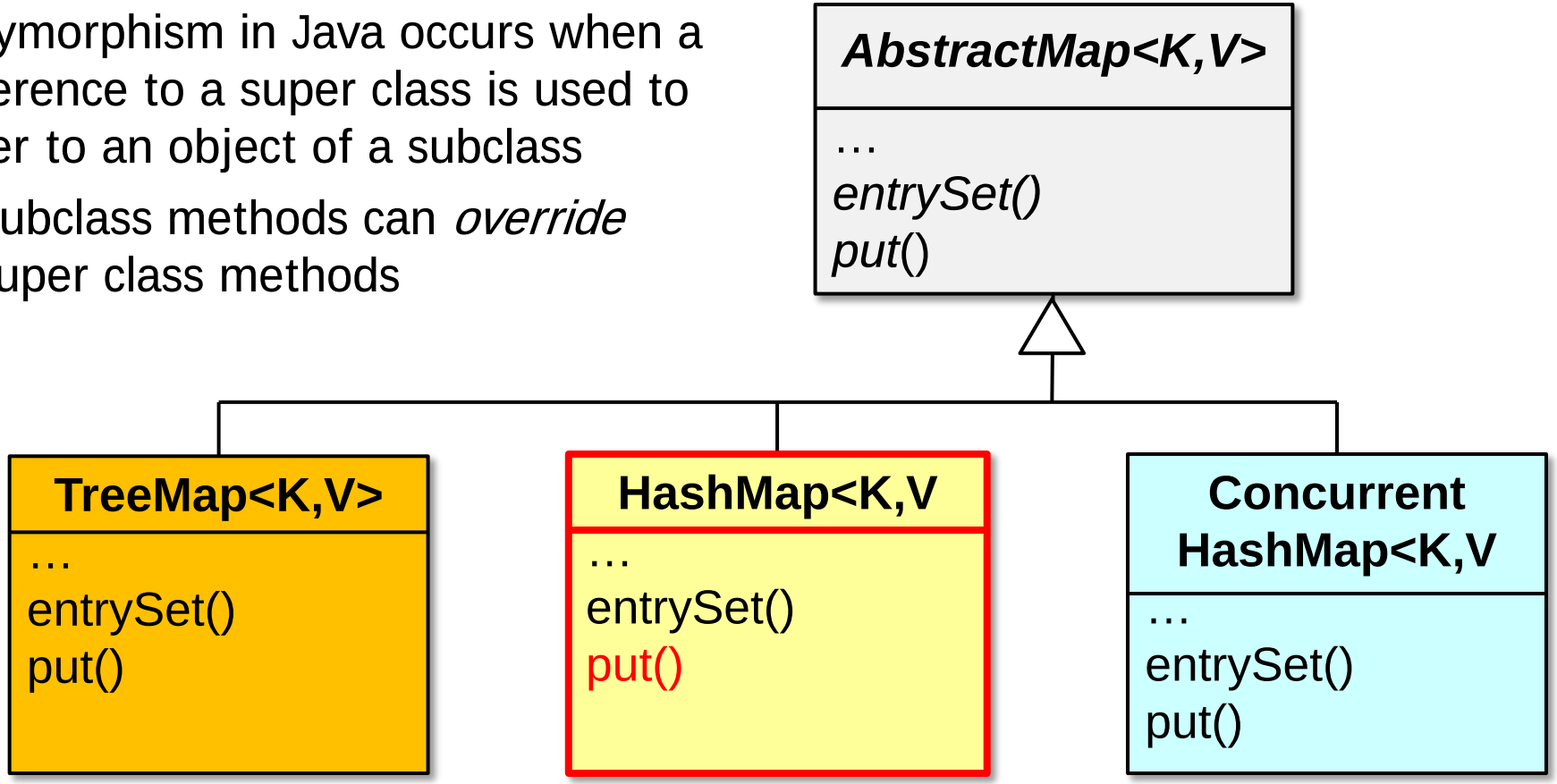
- Polymorphism in Java occurs when a reference to a super class is used to refer to an object of a subclass
  - Subclass methods can *override* super class methods



Subclasses of AbstractMap have different time & space tradeoffs

Overview of Java's Support for Polymorphism

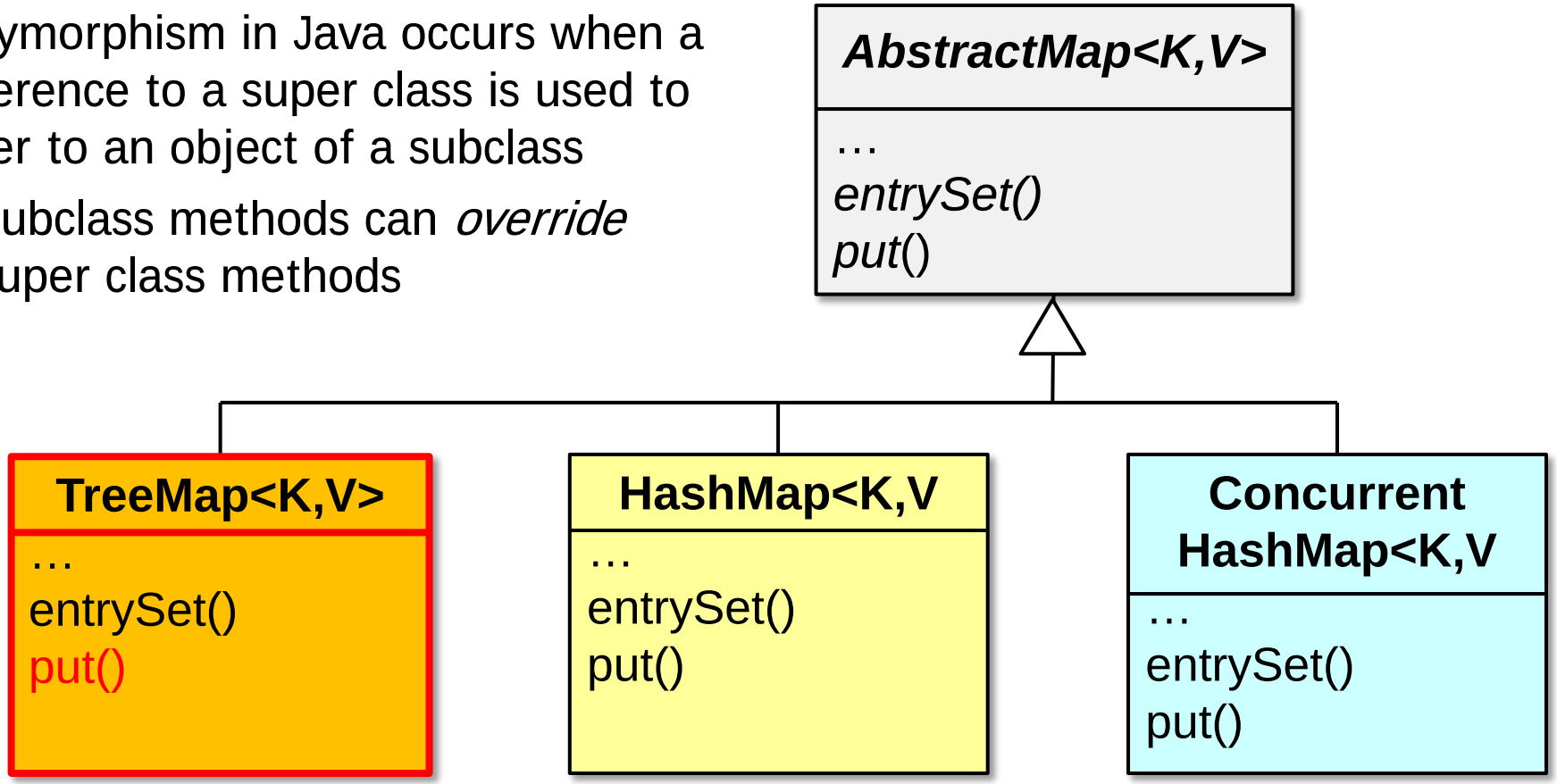
- Polymorphism in Java occurs when a reference to a super class is used to refer to an object of a subclass
  - Subclass methods can *override* super class methods



See developer.android.com/reference/java/util/HashMap.html

Overview of Java's Support for Polymorphism

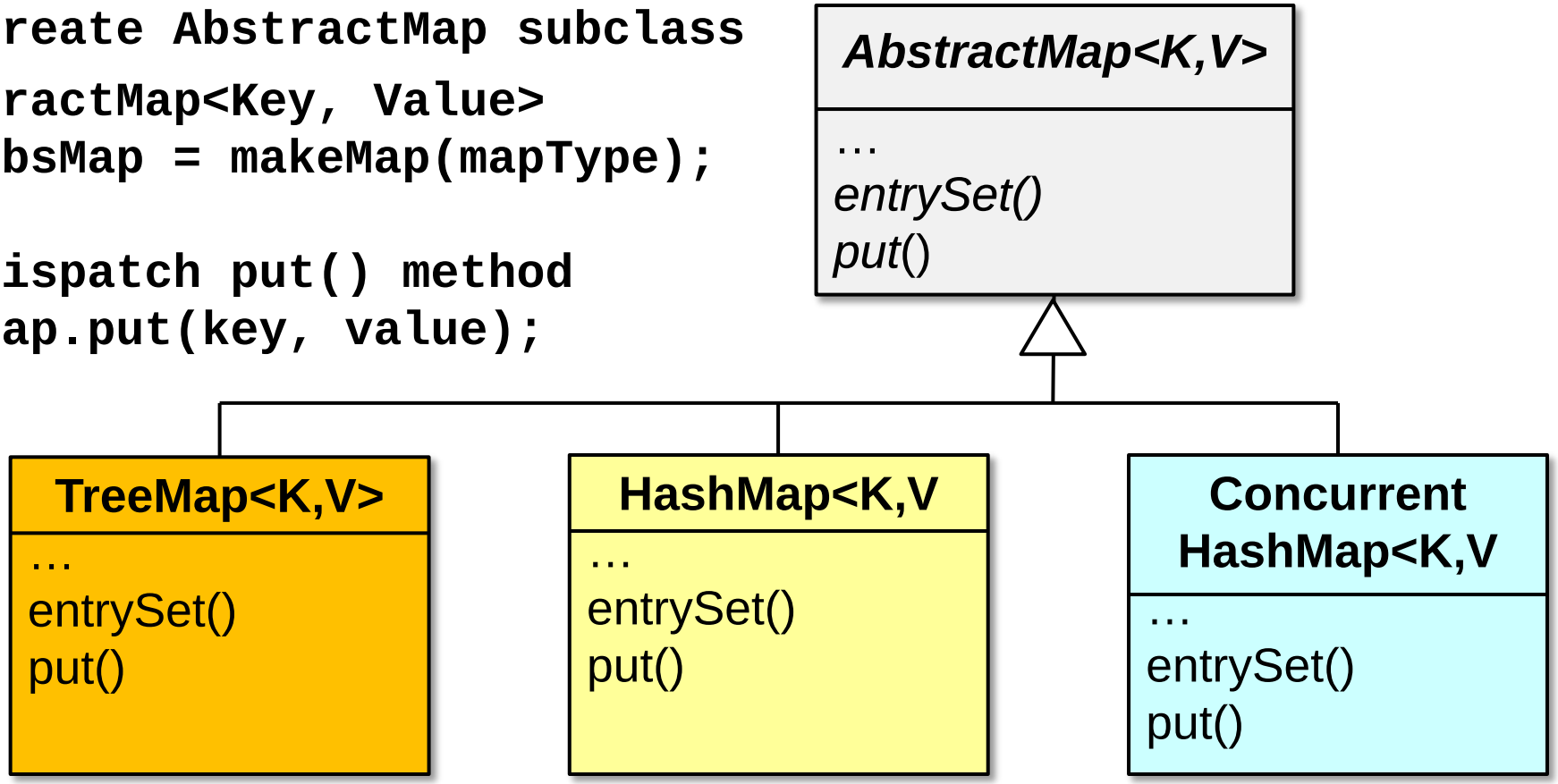
- Polymorphism in Java occurs when a reference to a super class is used to refer to an object of a subclass
  - Subclass methods can *override* super class methods



See developer.android.com/reference/java/util/TreeMap.html

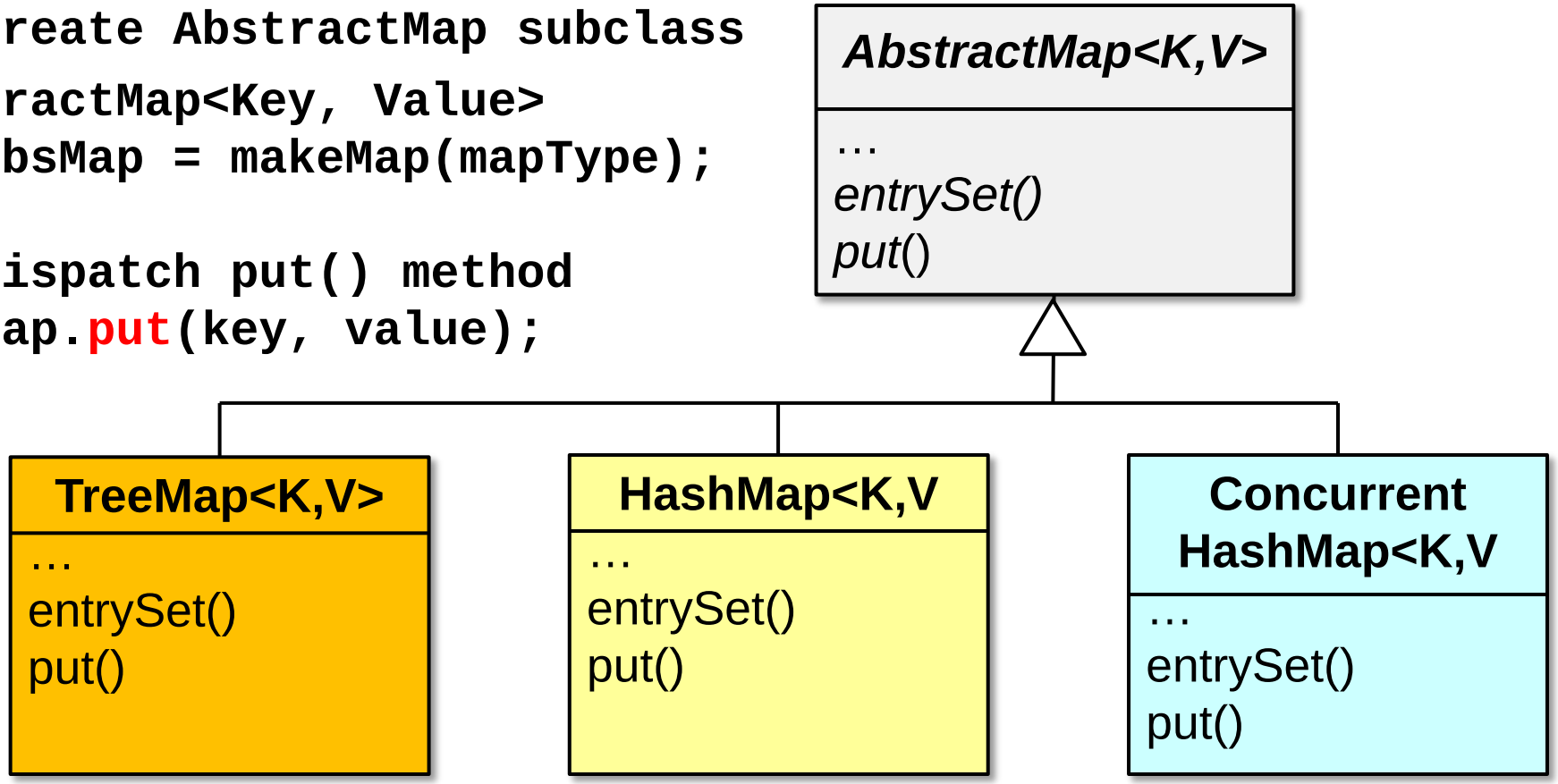
Overview of Java's Support for Polymorphism

```
// Create AbstractMap subclass
AbstractMap<Key, Value>
   absMap = makeMap(mapType);
...
// Dispatch put() method
absMap.put(key, value);
```



Overview of Java's Support for Polymorphism

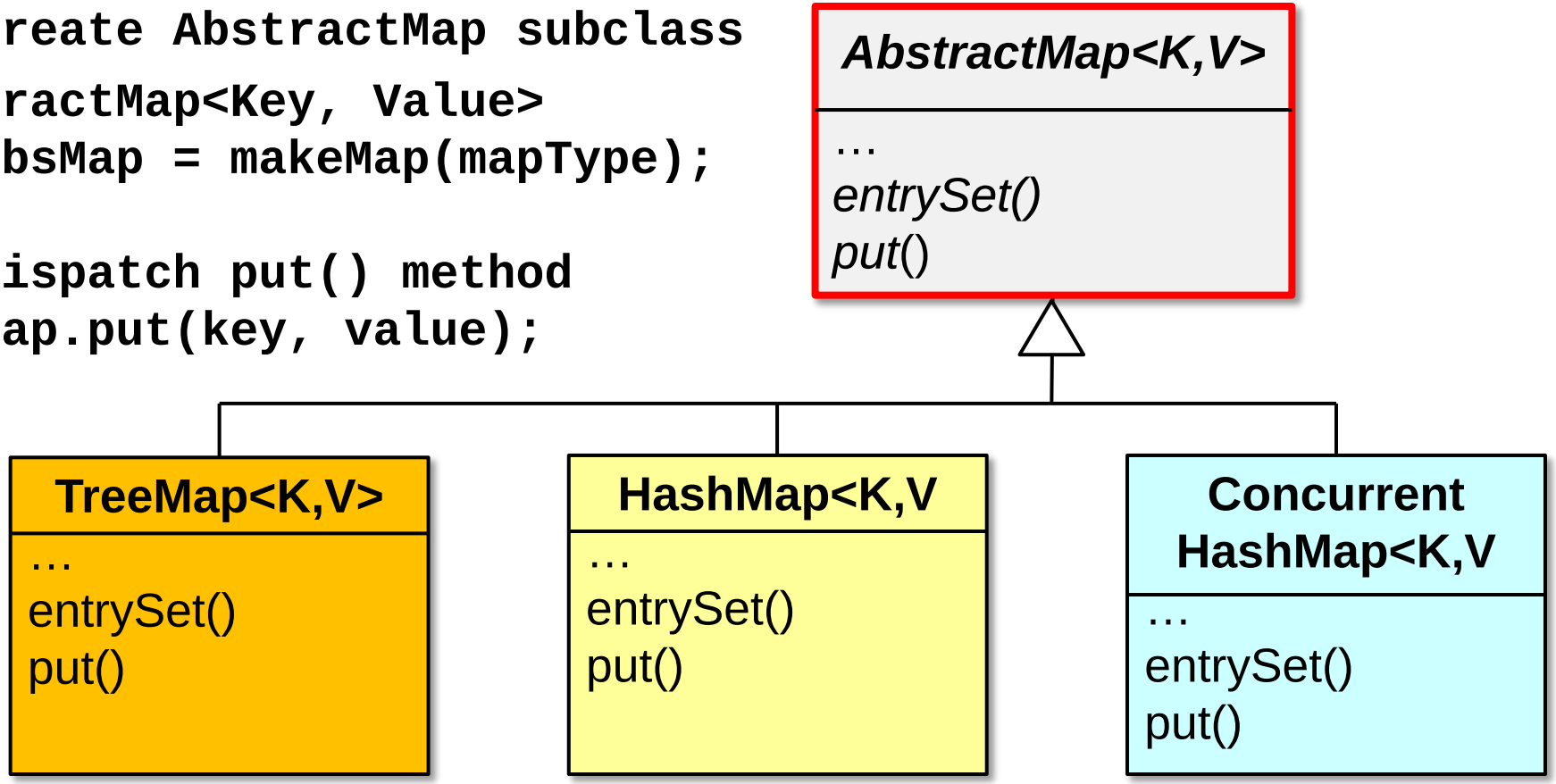
```
// Create AbstractMap subclass
AbstractMap<Key, Value>
   absMap = makeMap(mapType);
...
// Dispatch put() method
absMap.put(key, value);
```



The appropriate method is dispatched at runtime based on the subclass object

Overview of Java's Support for Polymorphism

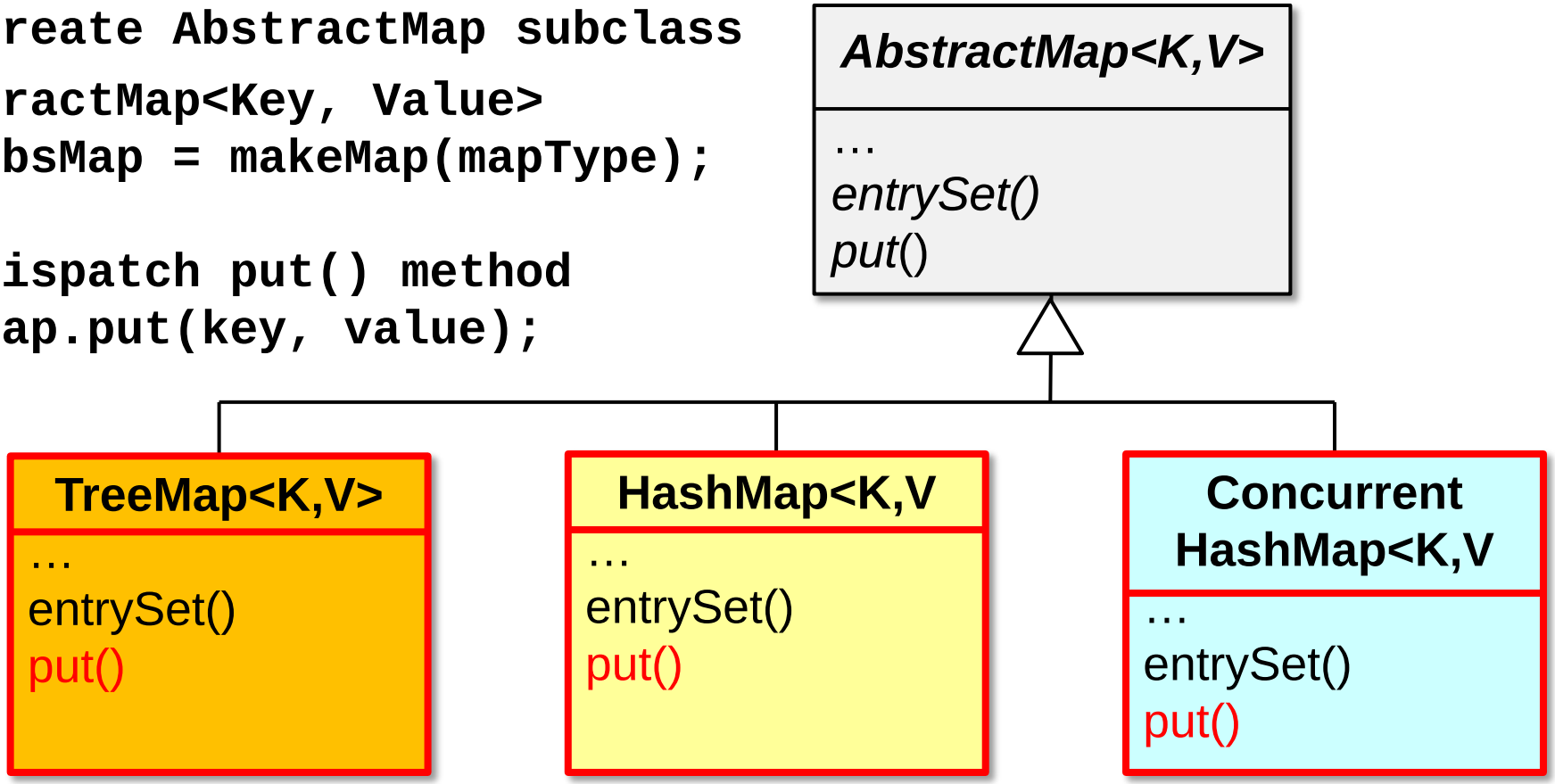
```
// Create AbstractMap subclass
AbstractMap<Key, Value>
   absMap = makeMap(mapType);
...
// Dispatch put() method
absMap.put(key, value);
```



The appropriate method is dispatched at runtime based on the subclass object

Overview of Java's Support for Polymorphism

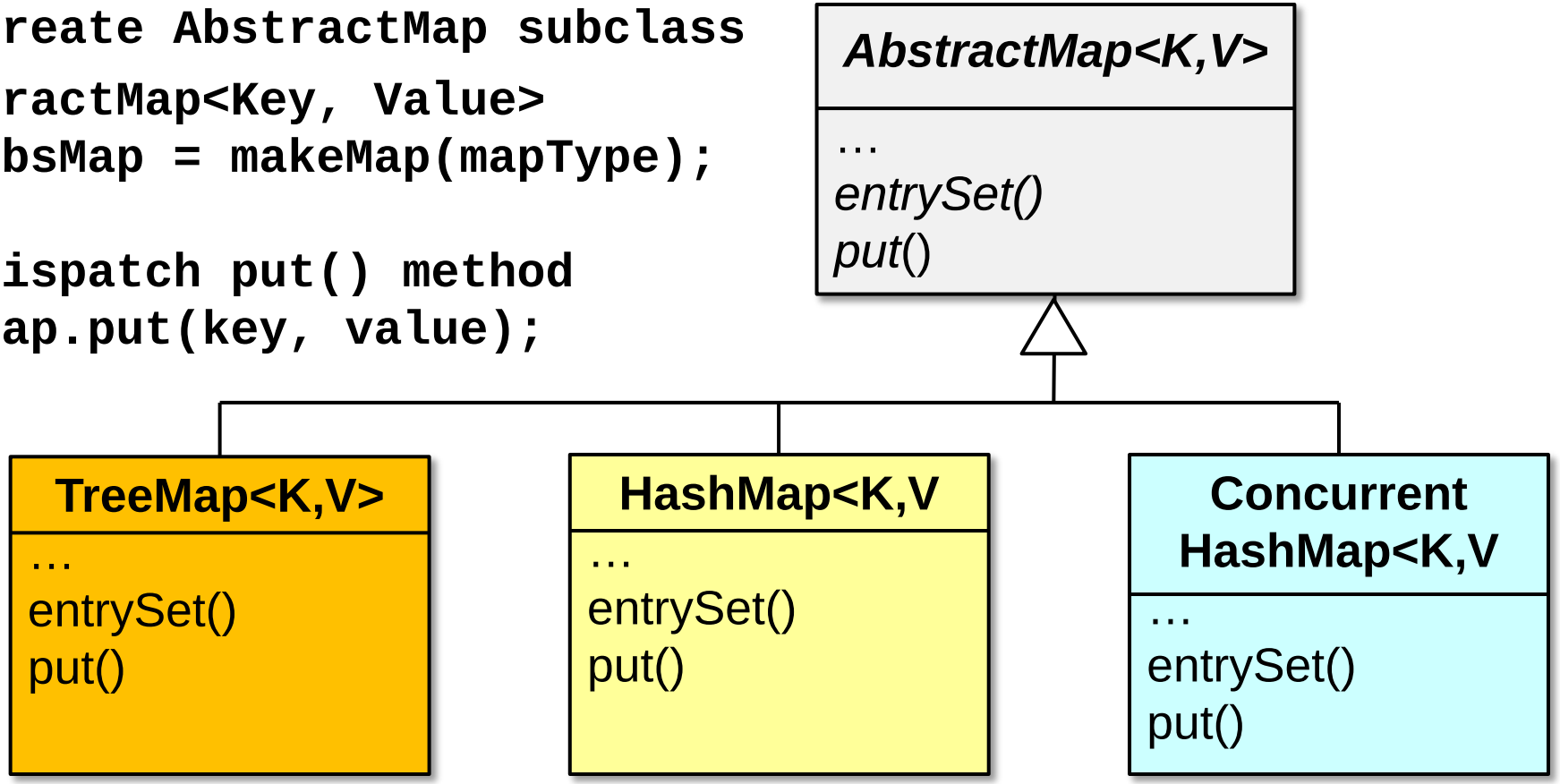
```
// Create AbstractMap subclass
AbstractMap<Key, Value>
   absMap = makeMap(mapType);
...
// Dispatch put() method
absMap.put(key, value);
```



The appropriate method is dispatched at runtime based on the subclass object

Overview of Java's Support for Polymorphism

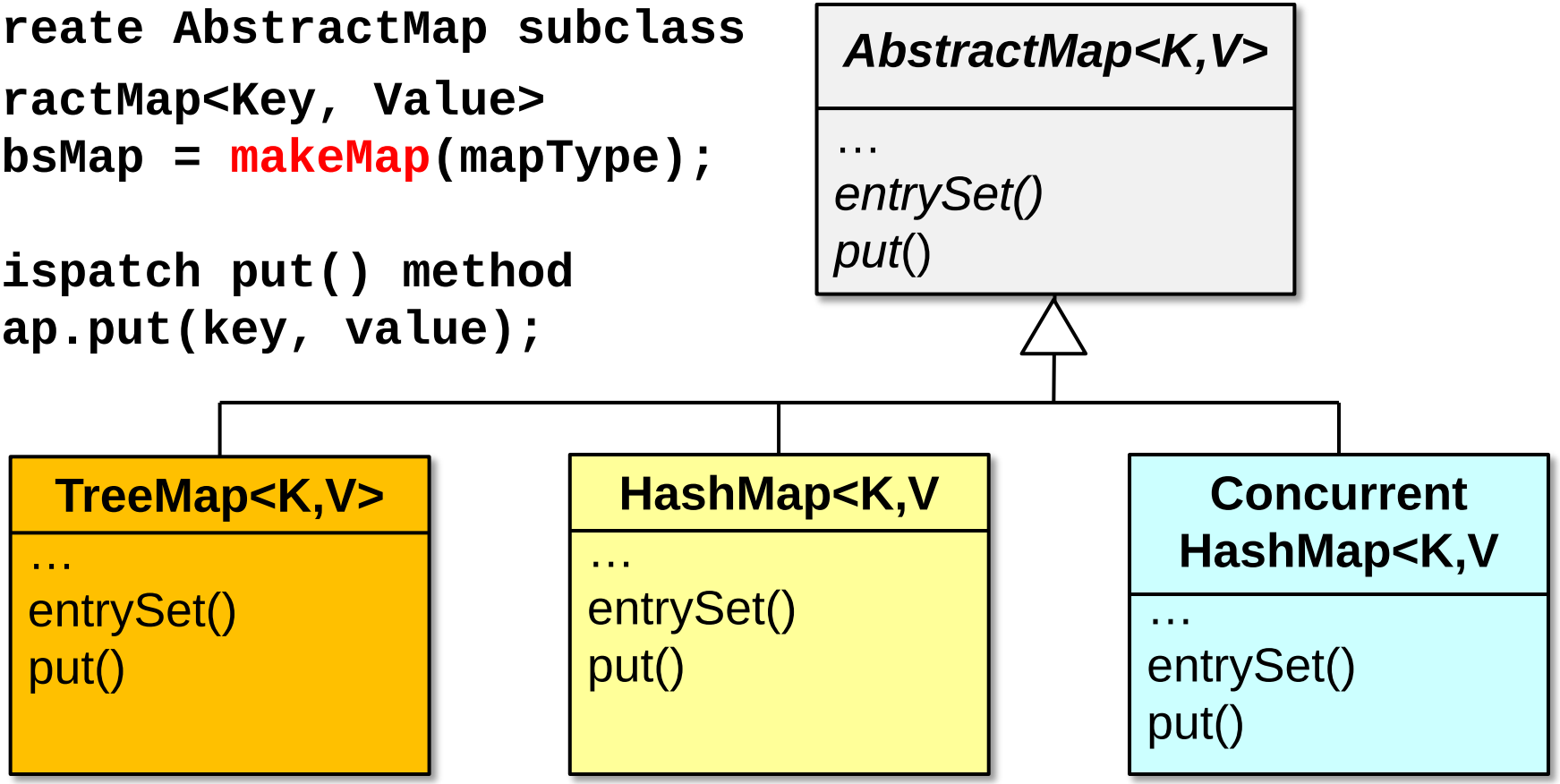
```
// Create AbstractMap subclass
AbstractMap<Key, Value>
   absMap = makeMap(mapType);
...
// Dispatch put() method
absMap.put(key, value);
```



See en.wikipedia.org/wiki/Factory_method_pattern & en.wikipedia.org/wiki/Open/closed_principle

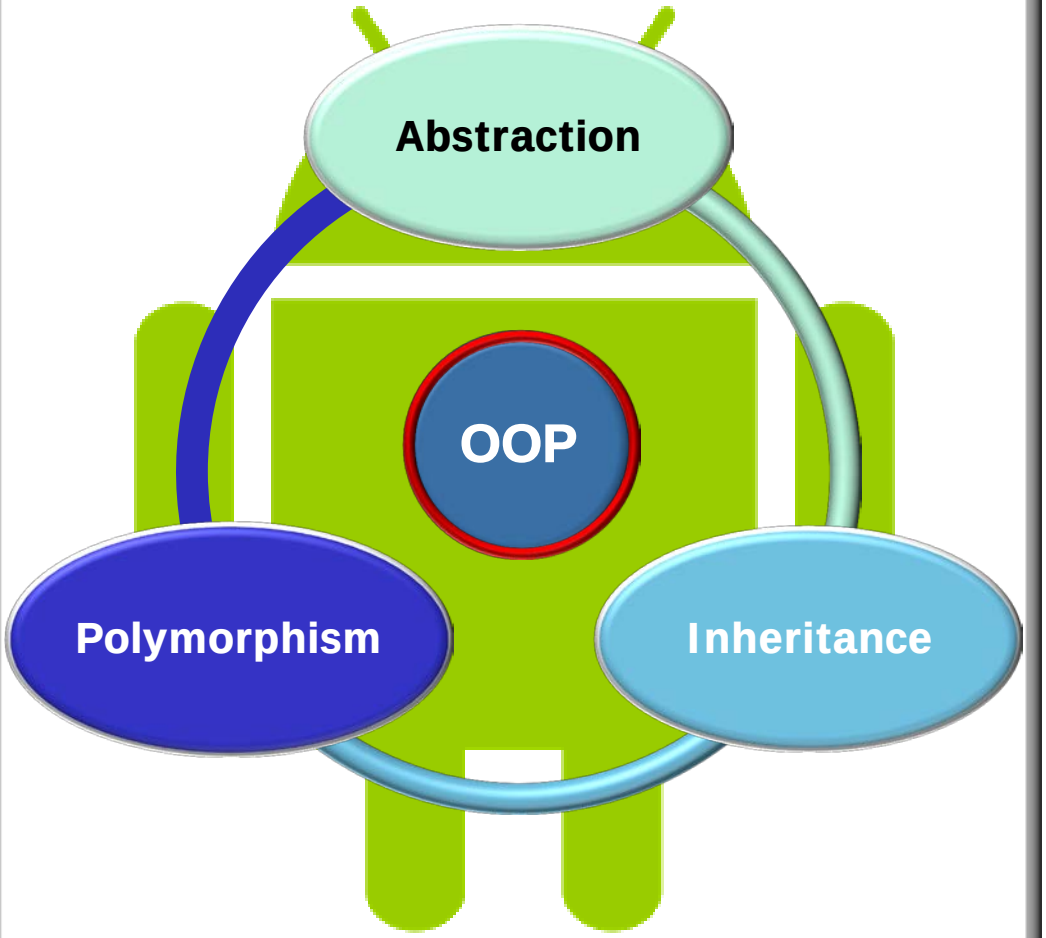
Overview of Java's Support for Polymorphism

```
// Create AbstractMap subclass
AbstractMap<Key, Value>
   absMap = makeMap(mapType);
...
// Dispatch put() method
absMap.put(key, value);
```

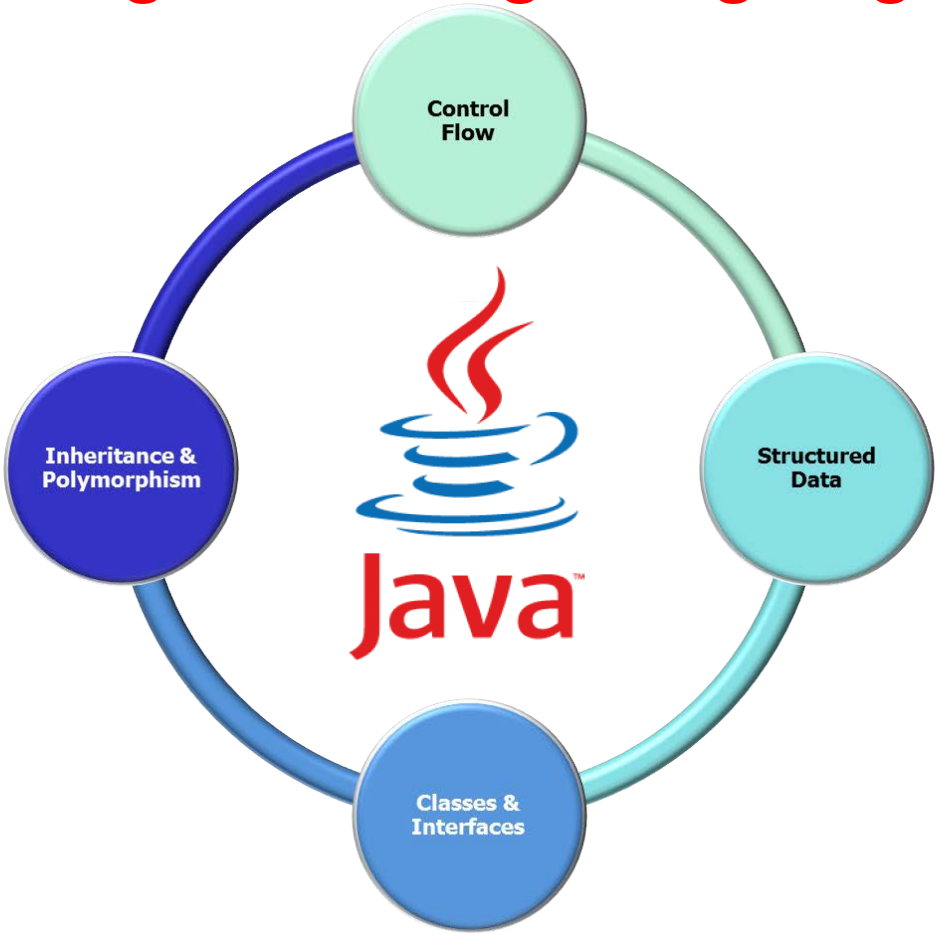


See en.wikipedia.org/wiki/Factory_method_pattern & en.wikipedia.org/wiki/Open/closed_principle

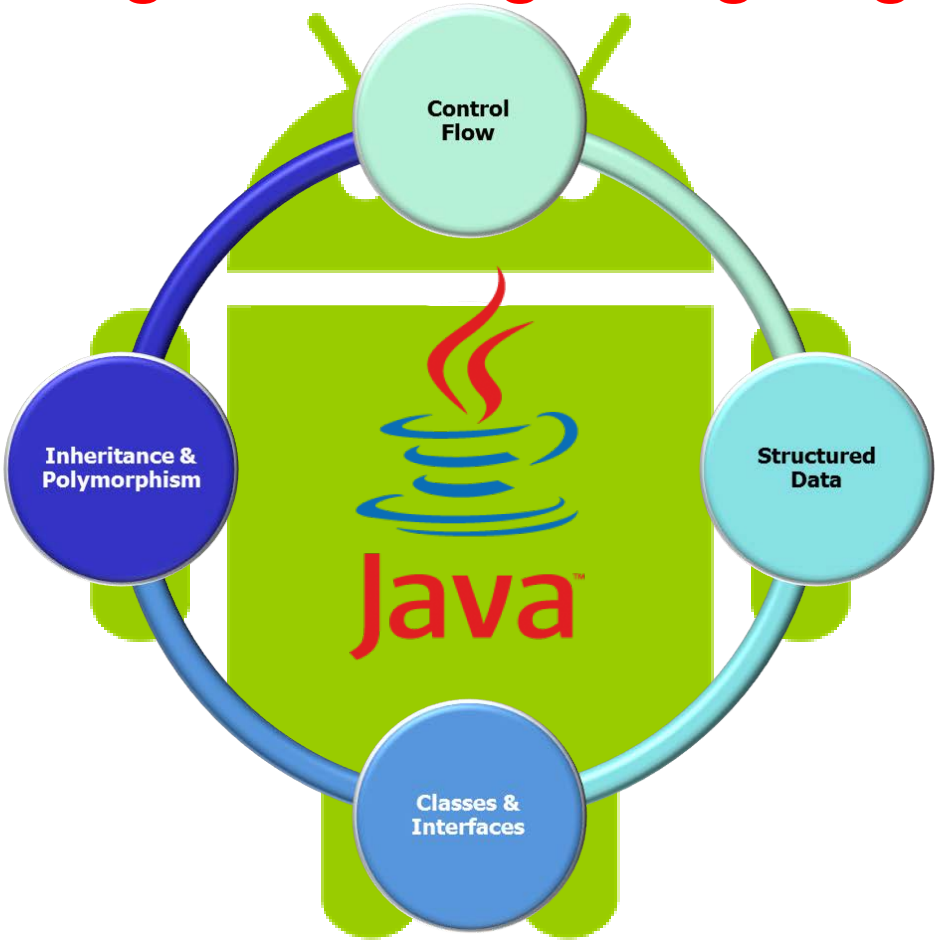
Overview of the Java Programming Language



Overview of the Java Programming Language



Overview of the Java Programming Language



Overview of the Java Programming Language

Learning Objectives

- Understand the key object-oriented (OO) concepts supported by Java
- Know the benefits that OO concepts provide developers of Java apps in Android
- Identify Java features that implement these OO concepts

Overview of the Java Programming Language

Learning Objectives

- Understand the key object-oriented (OO) concepts supported by Java
- Know the benefits that OO concepts provide developers of Java apps in Android
- Identify Java features that implement these OO concepts

Overview of the Java Programming Language

Learning Objectives

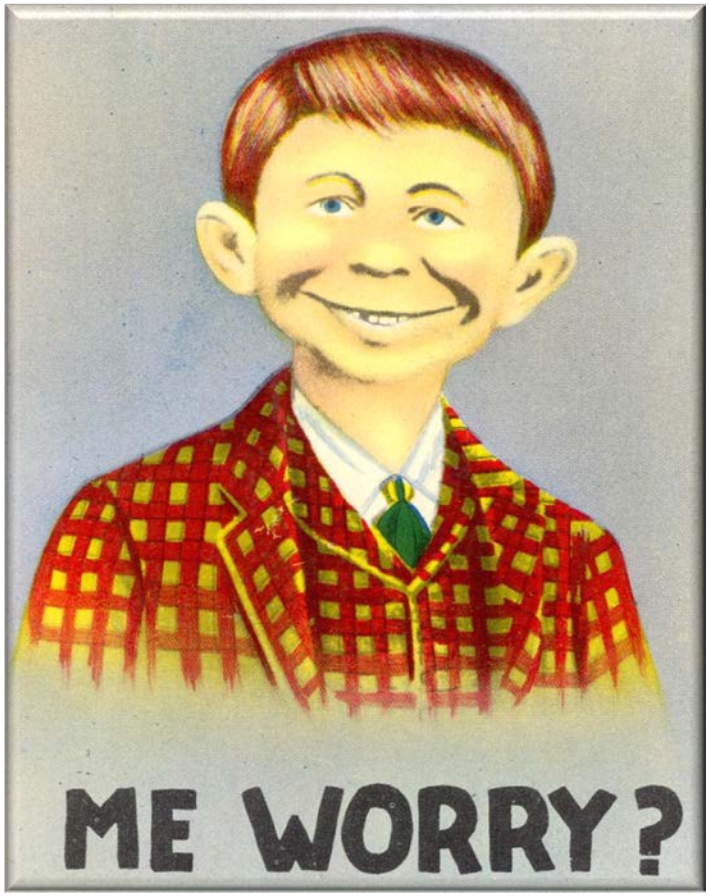
- Understand the key object-oriented (OO) concepts supported by Java
- Know the benefits that OO concepts provide developers of Java apps in Android
- Identify Java features that implement these OO concepts

Overview of the Java Programming Language

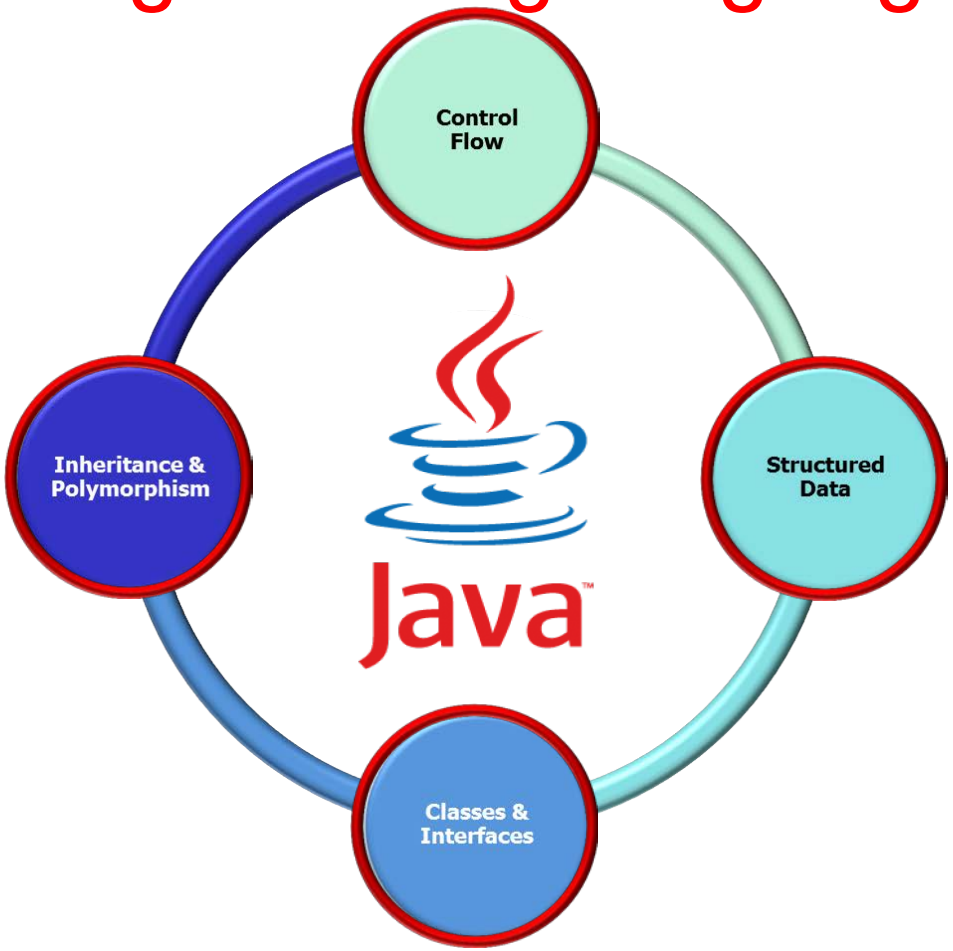
Learning Objectives

- Understand the key object-oriented (OO) concepts supported by Java
- Know the benefits that OO concepts provide developers of Java apps in Android
- Identify Java features that implement these OO concepts

Overview of the Java Programming Language

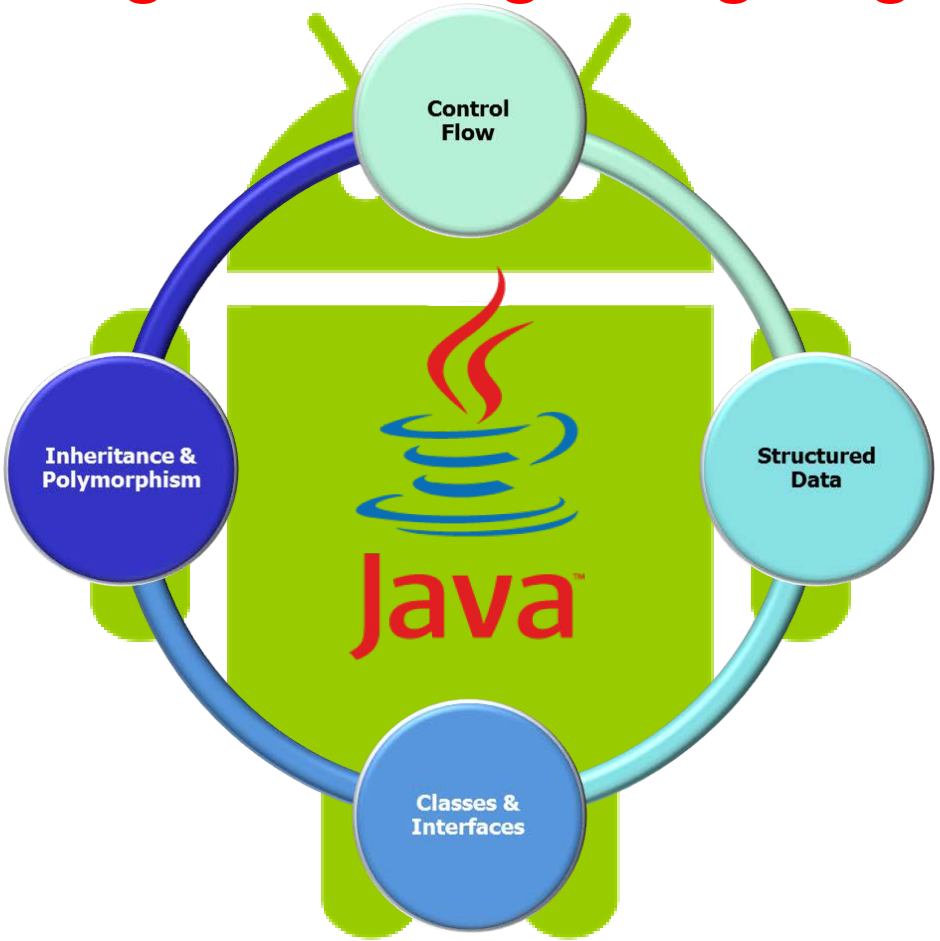


Overview of the Java Programming Language



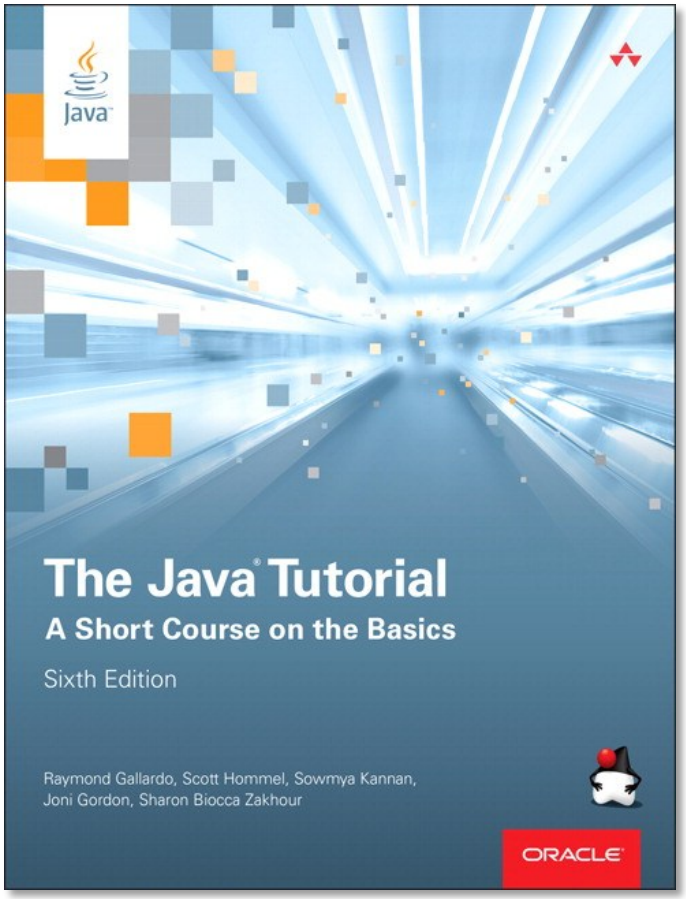
Other lessons examine Java in detail…

Overview of the Java Programming Language



...to learn to program Java for Android

Overview of the Java Programming Language



See download.oracle.com/javase/tutorial

Overview of the Java Programming Language

Package Index

These are the Android APIs. See all API classes.

java.util.concurrent	Utility classes commonly useful in concurrent programming.
java.util.concurrent.atomic	A small toolkit of classes that support lock-free thread-safe programming on single variables.
java.util.concurrent.locks	Interfaces and classes providing a framework for locking and waiting for conditions that is distinct from built-in synchronization and monitors.
java.util.jar	
java.util.logging	
java.util.prefs	
java.util.regex	
java.util.zip	
javax.crypto	This package provides the classes and interfaces for cryptographic applications implementing algorithms for encryption, decryption, or key agreement.
javax.crypto.interfaces	This package provides the interfaces needed to implement the Diffie-Hellman (DH) key agreement's algorithm as specified by PKCS#3.
javax.crypto.spec	This package provides the classes and interfaces needed to specify keys and parameter for encryption.
javax.microedition.khronos.egl	
javax.microedition.khronos.opengles	Provides a standard OpenGL interface.

See developer.android.com/reference/packages.html

Overview of the Java Programming Language