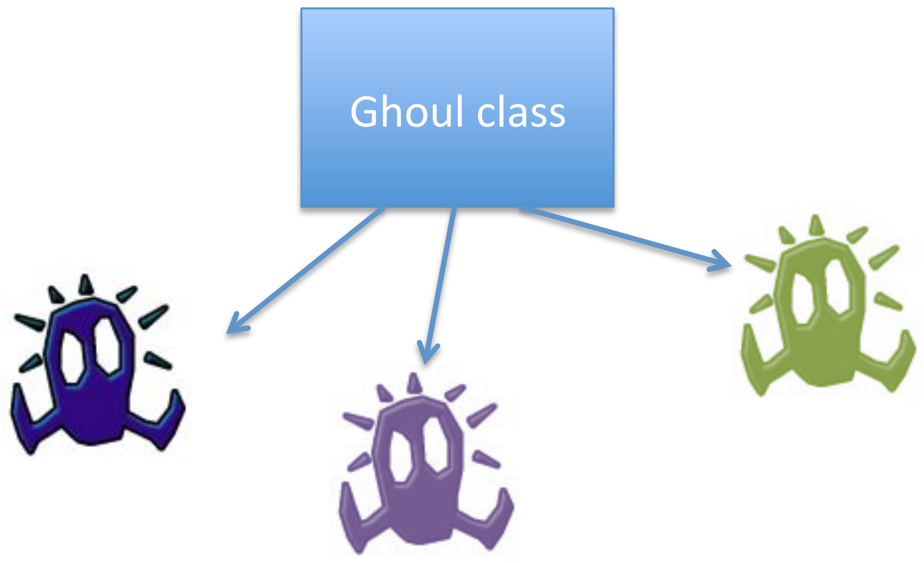
Classes that can be instantiated



Declaring a class as abstract

- creates “abstract types” whose implementations are incomplete or non-existent
- define & enforce a *protocol* that must be supported by subclasses

```
package java.util;

public abstract class AbstractMap<K,V> ... {
   ...
}

public class HashMap<K,V> extends AbstractMap<K,V>
{ ... }

public class TreeMap<K,V> extends AbstractMap<K,V>
{ ... }
```

AbstractMap is the super class for Map implementations

Declaring an abstract method

- An *abstract method* is a method declared without a body
- Subclasses must override these abstract methods and provide implementation details

```
public abstract void revive();
```

```
package java.util;

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

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

Abstract classes can not instantiate objects

- An abstract class can be extended, but not instantiated

```
// Fails
AbstractMap<Integer,String> map1 = new AbstractMap<>();
```

- A subclass that implements all abstract methods can be instantiated

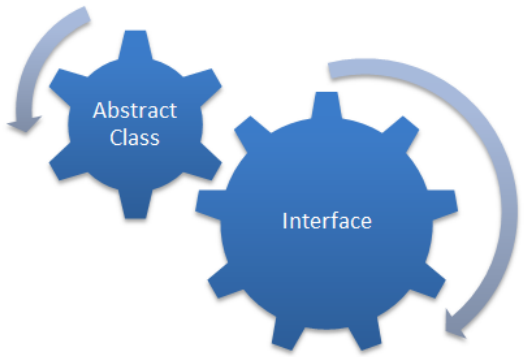
```
// Works
  AbstractMap<Integer, String> map2 = new HashMap<>();
```

Comparing Abstract Classes & Interfaces

Abstract classes are similar to interfaces

- They can't be instantiated
- They can contain methods declared without any implementation

However, abstract classes have additional capabilities



Comparing Abstract Classes & Interfaces

Abstract classes

- can define fields that are not static nor final
- can also define public, protected, & private concrete methods

Interfaces

- all fields must be public, static & final
- all methods must be public & not implemented

Comparing Abstract Classes & Interfaces

- A class can extend only one class

```
public class AtomicGhoul extends Ghoul{...
```

- A class can implement several interfaces

```
public class HazMatBox extends Box
  implements ShippingContainer,
  implements Cube{...
```

Overview of Java Nested Classes

A nested class is defined within an enclosing class

Overview of Java Nested Classes

Nested classes provide several benefits

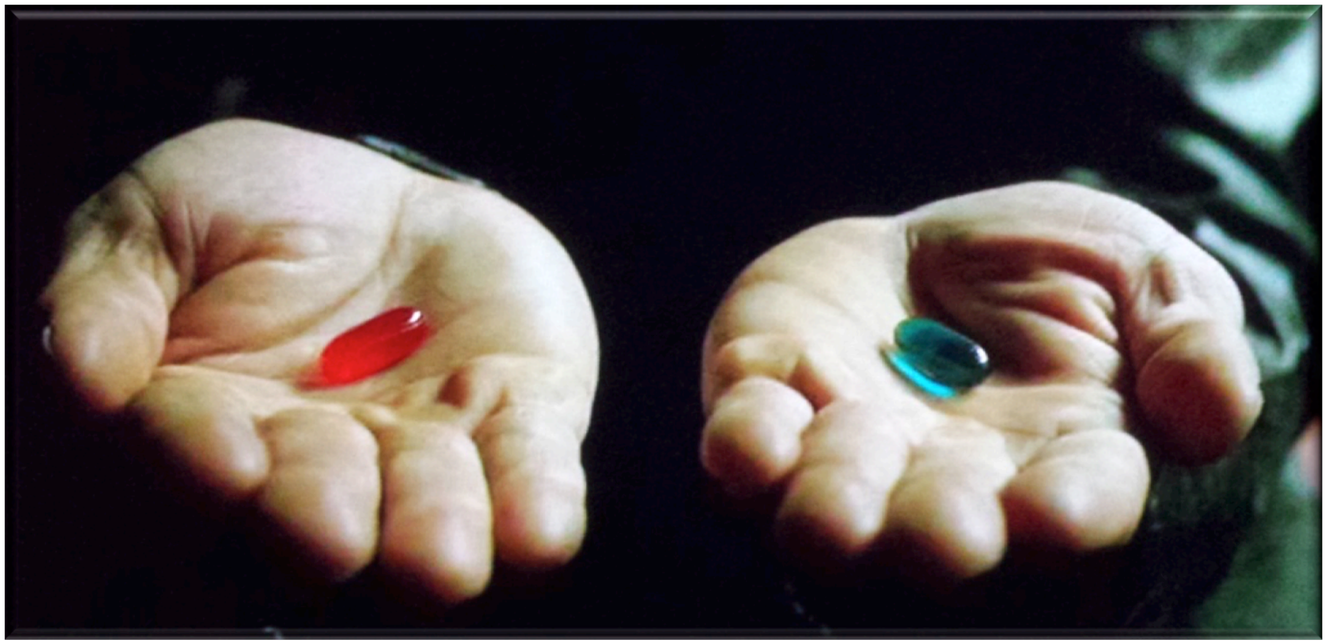
- Grouping together classes that are only used in one place
- Increasing encapsulation
- Enhancing maintainability

```
package java.util;

public class Vector<E> ... {
  ...
  public Spliterator<E>
  spliterator() {
     return new
     VectorSpliterator<>(...);
  }

  static final class
      VectorSpliterator<E>
      implements Spliterator<E>
  { ... }
  ...
```

There are two types of nested classes



Overview of Java Nested Classes

```
package java.util;

public class Vector<E> ... {
  ...
  int mCount;
  ...
  private class Itr
        implements Iterator<E>{
    int cursor;
    ...
    public boolean hasNext() {
      return cursor != mCount;
    }
```

An inner nested class *can* reference non-static instance methods & fields

Overview of Java Nested Classes

```
package java.util;

public class Vector<E> ... {
  ...

  static final class VectorSpliterator<E> implementsSpliterator<E> {
    ...
    private Object[] array;
    private int index;
    ...
  }
  ...
```

A static nested class *can't* reference non-static instance methods & fields