1

CS/ENGRD 2110 SPRING 2016

Lecture 7: Interfaces and Abstract Classes
http://courses.cs.cornell.edu/cs2110

Announcements

2

- Attendance for this week's recitation is mandatory!
- A2 is due Wednesday
- Get started on A3 – do one method a day

Overview

3

- Big Demo!
- Interfaces
- Abstract Classes
- Normal Classes vs. Abstract Classes vs. Interfaces

Interfaces

4

Interfaces cannot by **new**ed!

New keyword

```
/** A mutable collection of E values */
public interface Collection<E> {
  /** Return true if this collection contains elem*/
  boolean contains(Object elem);

  /** Ensure that this contains elem.
    * Return true if the collection is changed by this. */
  boolean add(E elem);
  …
}
```

Always public

No fields! No constructors!

No implementations!!!

Implementing Interfaces

5

Classes *implement* interfaces

```
/** A range of integers that always includes 0 */
public class IntRange implements Collection<Integer> {
  private int min = 0; // Represents the range min..max
  private int max = 0; // min <= max
  /** Return true if elem is an integer in the range. */
  public boolean contains(Object elem) {…}
  /** Minimally extend the range to include elem.
    * Return true if the range had to be extended. */
  public boolean add(Integer elem) {…}
}
```

Provides implementations for interface methods

Using Interfaces

6

```
/** Returns whether the collection contains every
  * integer between and including min and max.
  * Precondition: ints is not null */
public static boolean containsRange(
                    Collection<Integer> ints,
                    int min, int max) {
  for (int i = min; i <= max; i++)
    if (!ints.contains(i))
      return false;
  return true;
}
```

Interfaces are types

Works on any Collection!!! This includes your own DLinkedList from A3!

Because ints has type Collection<Integer>, you can use any method declared in the Collection<Integer> interface.

Extending Interfaces

7

Interfaces *extend* other interfaces

```
/** A mutable indexed list of E values */
public interface List<E> extends Collection<E> {
  int size(); // return size of the list
  E get(int index); // return elem at index
  E set(int index, E elem); // change elem at index
  boolean add(int index, E elem); // insert elem at index
  E remove(int index); // remove and return elem at index
  …
}
```

Implicitly includes all methods in Collection<E>

Abstract Classes

Outdated use of abstract classes! - see next slide on defaults

8

Abstract classes cannot be **newed**

Indicates that subclasses are responsible for providing the implementation

Only abstract classes can have abstract methods

```
/** Provides default implementations for list methods */
public abstract class AbstractList<E> implements List<E> {
  public abstract int size();
  public abstract E get(int index);
  public abstract E set(int index, E elem);
  public abstract boolean add(int index, E elem);
  public abstract E remove(int index);
  public boolean add(E elem) { return add(size(), elem); }
  public boolean contains(E elem) {
      for (int i = 0; i < size(); i++)
          if (!Objects.equals(elem, get(i)))
              return false;
      return true;
  }
  …
}
```

Defaults in Java 8

9

Indicates that the interface is providing a *default* implementation for this method

```
/** Provides default implementations for list methods */
public interface List<E> extends Collection<E> {
  int size();
  E get(int index);
  E set(int index, E elem);
  boolean add(int index, E elem);
  E remove(int index);
  default boolean add(E elem) { return add(size(), elem); }
  default boolean contains(E elem) {
      for (int i = 0; i < size(); i++)
          if (!Objects.equals(elem, get(i)))
              return false;
      return true;
  }
  …
}
```

Abstract Classes Revisited

10

Abstract class provides common fields and functionality

Abstract class leaves critical methods abstract for subclasses to implement

Subclasses provide case-dependent implementations

```
public abstract class IntExpression {
  private Integer value = null;
  public int evaluate() {
      if (value == null) value = eval();
      return value.intValue();
  }
  protected abstract int eval();
}
public class Zero extends IntExpression {
  protected int eval() { return 0; }
}
public class Sum extends IntExpression {
  protected IntExpression left, right;
  public Sum(…) {…}
  protected int eval() { return left.eval() + right.eval(); }
}
```

Comparison

11

Normal Classes	Abstract Classes	Interfaces	Features
✓	✓	✓	can be used as types and in casts
✓	⊘	⊘	can be **newed**
✓	✓	⊘	have constructors
✓	✓	⊘	can have fields
✓	✓	✓ (Java 8)	can provide method implementations
✓	✓	⊘	can have non-public methods
⊘	✓	✓	can have abstract methods
⊘	⊘	✓	can be inherited multiply