# Implementing Collections

## Lecture 7

# Menu

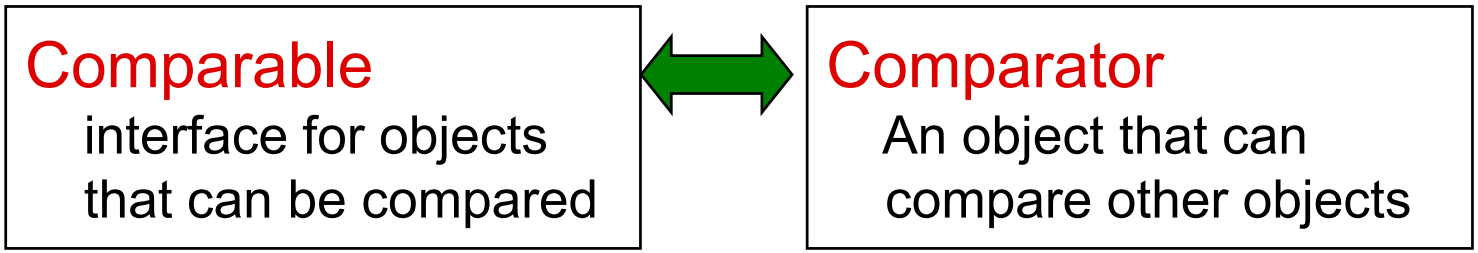
- Comparators
- Exceptions
- Implementing Collections:
  - Interfaces, Classes

# Sorting with Comparators

- Suppose we need two different sorting orders at different times?
- Collections.sort(…) has two forms:
  - Sort by the natural order

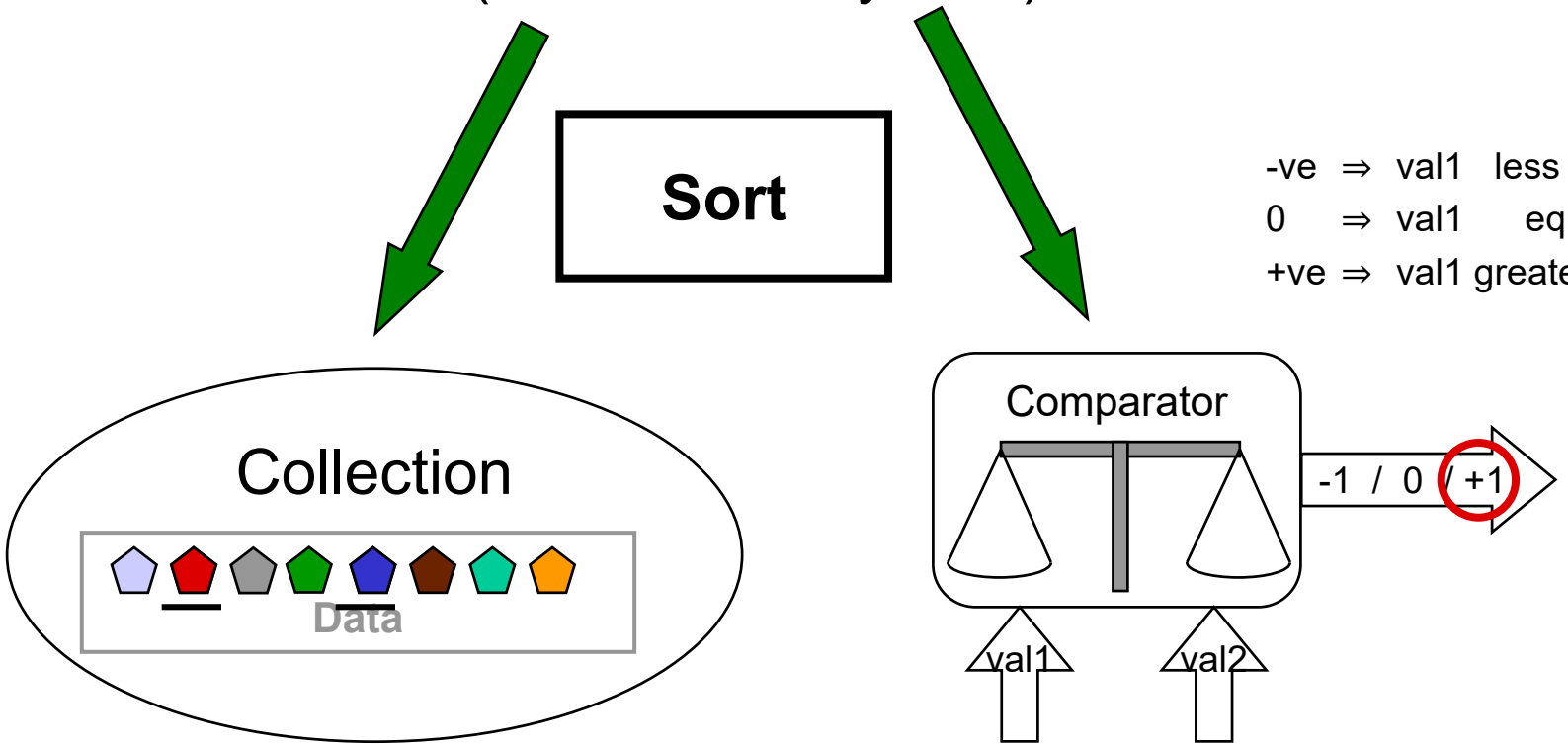
    Collections.sort(todoList)
    - the values in todoList must be comparable
  - Sort according to a specified order:

    Collections.sort(crowd, faceByArea)
    - faceByArea is a Comparator object for comparing the values in crowd.



# Sorting with Comparators

- Collections.sort(crowd, faceByArea);



# Comparators

- Comparator <T> is an Interface
- Requires
  - **public** int compare(T o1, T o2);
    - → –ve if o1 ordered before o2
    - → 0 if o1 equals o2
    - → +ve if o1 ordered after o2

```
/** Compares faces by the position of their top edge */
private class TopToBotComparator implements Comparator<Face>{
    public int compare(Face f1, Face f2){
        return (f1.getTop() - f2.getTop());
    }
}
```

# Using Multiple Comparators

```
if (button.equals("SmallToBig")){
      Collections.sort(crowd);    // use the "natural ordering" on Faces.
      render();
}
else if (button.equals("BigToSmall")){
      Collections.sort(crowd, new BigToSmallComparator());
      render();
}
 else if (button.equals("LeftToRight")){
      Collections.sort(crowd, new LfToRtComparator());
      render();
}
 else if (button.equals("TopToBottom")){
      Collections.sort(crowd, new TopToBotComparator());
      render();
}
```

# Exceptions

- What should a method do when something goes wrong?
  - Throw an exception!
  - An exception is an event that
    - occurs during the execution of a program
    - disrupts the normal flow of instructions
- An Exception is a "non-local exit" (what does that mean?)
  - Error of some kind, eg
    - file doesn't exist,
    - dividing by zero
    - calling a method on the null object
    - calling an undefined method
  - An exceptional situation, eg
    - text in file doesn't have the expected format
    - user enters an unexpected answer
- How do you deal with Exceptions?
  - **throwing exceptions**
  - **handling (catching) exceptions**

May be thrown by the JVM when executing code.

May be thrown by code from the java library

May be deliberately thrown by your code.

# Catching exceptions

- exceptions thrown in a **try ... catch** can be "caught":

```
try {
        …  code that might throw an exception
}
catch (⟨ExceptionType1⟩ e1) { …actions to do in this case….}
catch (⟨ExceptionType2⟩ e2) { …actions to do in this case….}
catch (⟨ExceptionType3⟩ e3) {      System.out.println(e.getMessage());
                                   e.printStackTrace(); }
```

Exception object, containing information about the state

Informative action for an error condition that the program can't deal with itself

- Meaning:
  - **If an exception is thrown, jump to catch claus**
  - **Match exception type against each catch clau**
  - **If caught, perform the actions, and jump to code after all the catch clauses**

  The actions may use information in the exception object.

# Catching exceptions

```
String filename = "/nosuchdir/myfilename";
try {
    // Create the file
    new File(filename).createNewFile();
} catch (IOException e) {
    // Print out the exception that occurred
    System.out.println("Unable to create "+filename+": "+e.getMessage());
}
// Execution continues here after the IOException handler is executed
```

Here's the output:
Unable to create /nosuchdir/myfilename: The system cannot find the path specified

# Types of Exceptions

- Lots of kinds of exceptions

**Exceptions**

AclNotFoundException, ActivationException, AlreadyBoundException, ApplicationException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CertificateException, ClassNotFoundException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSException, IllegalAccessException, IllegalClassFormatException, InstantiationException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, InvocationTargetException, **IOException**, JMException, LastOwnerException, LineUnavailableException, MidiUnavailableException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NoSuchFieldException, **NoSuchMethodException**, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, RefreshFailedException, RemarshalException, **RuntimeException**, SAXException, ServerNotActiveException, SQLException, TimeoutExce
Unsupported
Unsupported
XPathExcep

RunTimeExceptions:

AnnotationTypeMismatchException, ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DOMException, EmptyStackException, EnumConstantNotPresentException, EventException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IncompleteAnnotationException, **IndexOutOfBoundsException**, JMRuntimeException, LSException, MalformedParameterizedTypeException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, **NullPointerException**, ProfileDataException, ProviderException, RasterFormatException, RejectedExecutionException, SecurityException, SystemException, TypeNotPresentException, UndeclaredThrowableException, UnmodifiableSetException, **UnsupportedOperationException**

# Types of Exceptions

- Lots of exceptions

- **RuntimeExceptions don't have to be handled**:
  - An uncaught RuntimeException will result in an error message
  - You can catch them if you want.

- **Other Exceptions must be handled**:
  - eg IOException

    (which is why we always used a ***try…catch*** when opening files).

- An exception object contains information:

  the state of the execution stack

  any additional information specific to the exception

# Throwing exceptions

- The ***throw*** statement causes an exception.
  - eg, a drawing file has a line with an unknown shape:

```
if (name.equals("oval"))                shapes.add(new Oval(file));
else if (name.equals("rectangle"))      shapes.add(new Rectangle(file));
else if (name.equals("line"))           shapes.add(new Line(file));
            :
else if (name.equals("polyline"))       shapes.add(new PolyLine(file));
else
    throw new RuntimeException("Unknown shape in drawing file");
render();
```

General RuntimeException object.
- Could use a more specific one
- Could define our own type of exception

  - eg, defining a method that shouldn't be
    such as adding an element to an immutable List

```
public void add(E element){
    throws new UnsupportedOperationException("Immutable List");
}
```

# Throwing exceptions

```
public Object pop() {
        Object obj;
        if (size == 0) {
                throw new EmptyStackException();
        }
… …
}
```

# Implementing Collections

- Part 1: using collections
  - ✔ Different types of collections
  - ✔ Alternative data structures and algorithms
    - Cost and efficiency
    - Testing
    - Binary search algorithm
    - Sorting algorithms
- Part 2: implementing collections
  - ✔ What it means to implement a Collection Interface
  - ✔ The Java collections library
  - ✔ Interfaces for List, Set
  - ✔ Using “for each” loop, Iterable, and Iterators
  - ✔ Using Sort, Comparable, and Comparators
  - ✔ Throwing exceptions

# Interfaces and Classes

- Interface
  - specifies **type**
  - defines **method headers** only

- List <*E* >
  - Specifies sequence of *E*
  - size, $add_1$, get, set, $remove_1$
  - $add_2$, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, indexOf, lastIndexOf, iterator, listIterator, $remove_2$, removeAll,retainAll, subList, toArray.
- ArrayList <*E* >
  - implements List <*E* >
  - defines fields with array of <*E* > and count
  - defines size(), add(), …
  - defines …
  - defines constructors

- Class
  - implements Interface
  - defines **fields** for the data structures to hold the data.
  - defines **method bodies**
  - defines **constructors**

# Defining List: Type Variables

```
public interface List <E> extends Collection<E>{
    public int size ();
    public E get (int index);
    public E set (int index, E elem);
    public boolean add (E elem);
    public boolean add (int index, E elem);
    public E remove (int index);
    public void remove (Object ob);
    public Iterator<E> iterator();
    public void clear();
    public boolean contains (Object ob);
    public int indexOf (Object ob);
    public boolean addAll (Collection<E> c);
        :
```

E is a type variable

E will be bound to an actual type when a list is declared:

```
private List <String> items;
```

# Defining ArrayList

- Design the data structures to store the values
  - array of items
  - count

- Define the fields and constructors

```
public class ArrayList <E> implements List<E> {
    private E [ ] data;
    private int count;
```

- Define all the methods specified in the List interface

# Q&A

- A method will do what when something goes wrong?
- An exception provides a local exit when something goes wrong. (T or F)
- Name 3 cases when an exception can be thrown.
- What do we do with exceptions?
- Exceptions can be caught using what Java statement?
- Does exception have a type?
- After the exception handler finishes its work, what will the program do next?
- Can exception handler use information in the exception object?
- What does "**e.getMessage**()" do where **e** is an exception object?

# Q&A

- Name 3 common Java exceptions.
- RuntimeExceptions doesn't have to be handled. (T or F)
- IOException doesn't have to be handled. (T or F)
- Which Java statement can cause an exception?
- What happens when we try adding an element to an immutable List?
- Does a Java Interface provide a 'constructor'?
- ArrayList implements which Interface?
- **public interface** List <E> extends which Java Interface?

# Summary

- Comparators
- Exceptions
- Implementing Collections:
  - Interfaces, Classes