Department of Computer Science
University of Bristol

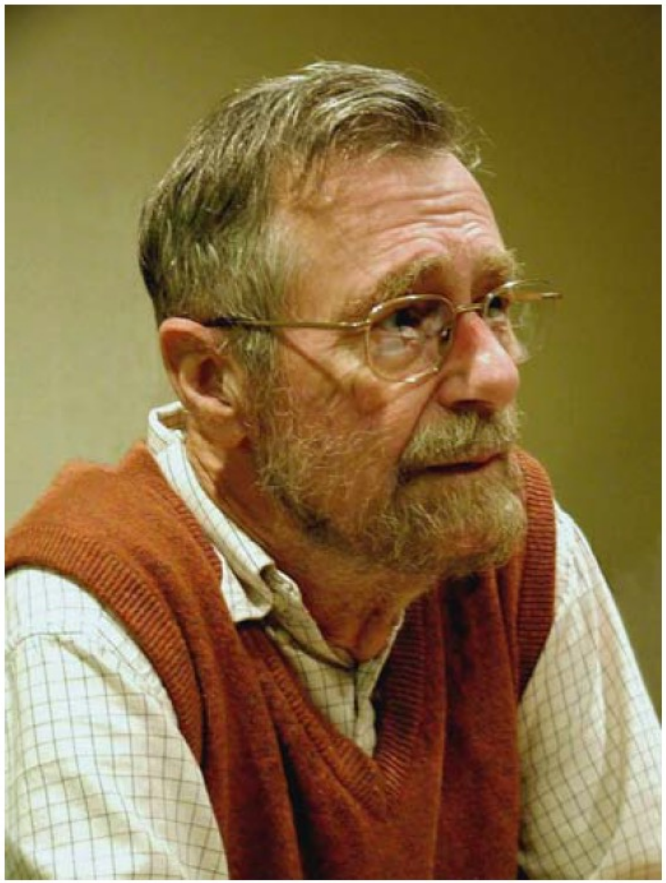
COMSM0086 – Object-Oriented Programming with Java

Iterator Design Pattern

Simon Lock | simon.lock@bristol.ac.uk
Sion Hanunna | sh1670@bristol.ac.uk

This thing …

```
for(String s : strings){
  System.out.println (s);
}
```



"…simplicity and elegance are unpopular because they require hard work and discipline to be achieved, and education to be appreciated..."

--- *E. Dijkstra*

"All problems in computer science can be solved by another level of indirection"

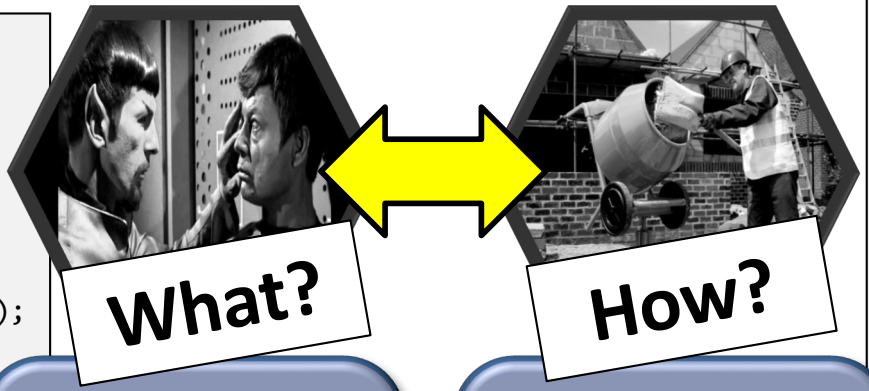
--- *Butler Lampson*

Interfaces
vs.
Concrete Implementations

Interfaces vs. Implementations

- the role of an **`interface`** (e.g. a **`Set`**) is to provide a contract
- any particular concrete implementation (e.g. **`ArraySet`**) has to fullfill it
- an **`interface`** does not force a particular way of realising this contract

```
interface Set<X> {

  public void insert(X x);

  public void delete(X x);

  public void empty();

  public boolean contains(X x);

  public int size();

}
```



interface Set provides a representation-independent contract, which all concrete implementations have to realise

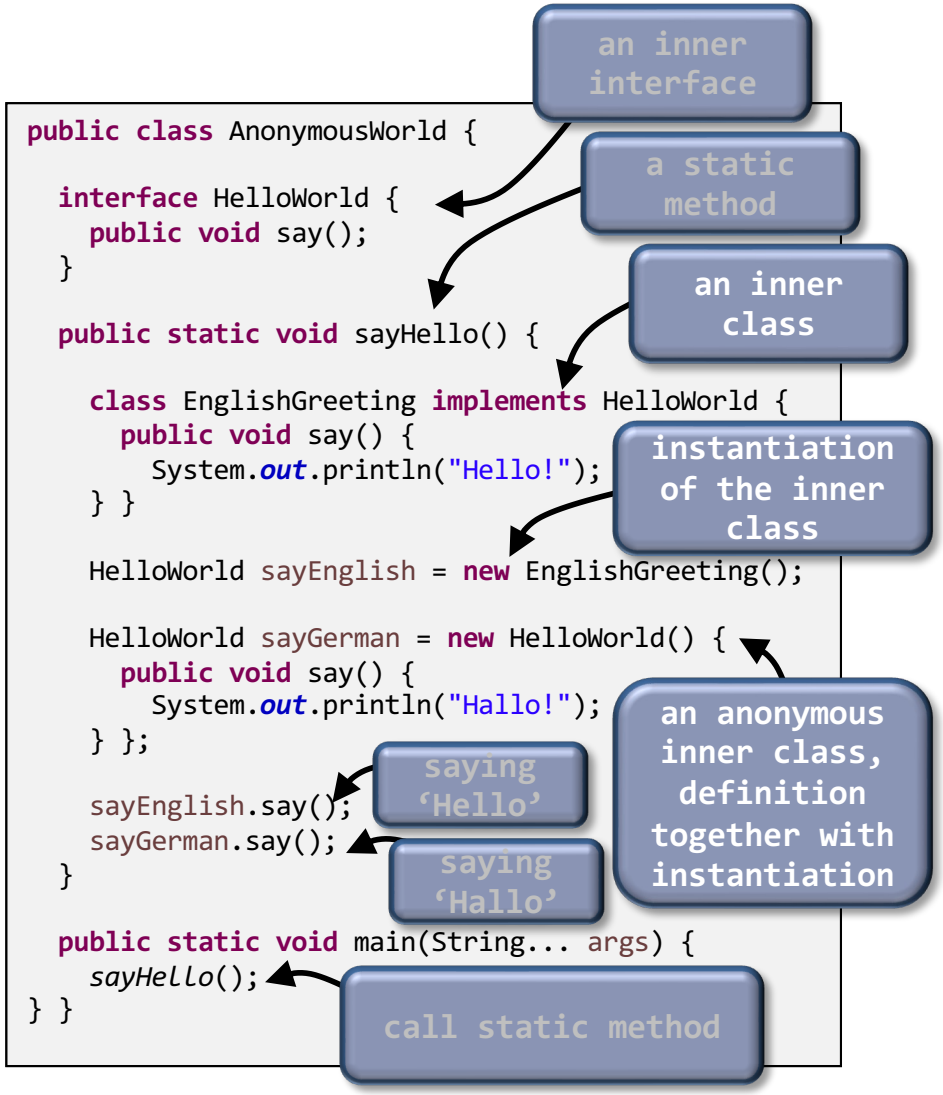
ArraySet implements all methods demanded by the interface Set and specifies a particular, concrete representation of all state and behaviour required

```
class ArraySet<X> implements Set<X> {
  protected X[] values;
  protected int size;
  private final int N = 100;

  public ArraySet() {
    values = (X[]) new Object[N];
    size = 0; }

  @Override
  public void insert(X x) {
    assert(size<100);
    assert(!contains(x));
    values[size] = x;
    size = size + 1; }

  @Override
  public void delete(X x) {
    assert(contains(x));
    for (int i=0; i < size; i = i+1) {
      if (values[i].equals(x)) {
        values[i] = values[size-1];
        size = size - 1;
        break;
  } } }

  @Override
  public boolean contains(X x) {
    boolean contains = false;
    for (X value : values) {
      if (value.equals(x)) {
        contains = true;
        break;
    } }
    return contains; }

  @Override
  public int size() {
    return size; }

  @Override
  public void empty() {
    size = 0;
} }
```

Inner Classes

Inner Classes and Anonymous Classes

- **inner classes** (or inner interfaces too) are defined within another class (the outer class)
- **anonymous (inner) classes** are defined and instantiated in a single expression using **new**, where the anonymous class definition itself is actually an expression
- it can be included as part of a larger expression, such as a method call
- inner classes are often local helper classes, whilst anonymous classes are often use-once classes without an explicit handle to the code that defines it



Iterators

This thing …

```
for(String s : strings){
  System.out.println (s);
}
```

The Concept of Iterators

- various object structures hold elements: e.g. sets, arrays, lists, trees (e.g. **`ArraySet`** on left)
- we often want to be able to iterate over **all** the elements **independent** of the structure
- Java has an **`Iterator`** interface to do this (see the JavaDocs for all details)
- classes need to implement **`Iterable`** to be iterated over using the **`:`** notation, the interface demands to be able to get hold of an **`Iterator`** object to drive it

```
class ArraySet<X> implements Set<X> {
  protected X[] values;
  protected int size;
  private final int N = 100;

  public ArraySet() {
    values = (X[]) new Object[N];
    size = 0; }

  @Override
  public void insert(X x) {
    ...
    values[size] = x;
    size = size + 1; }

  @Override
  public void delete(X x) {
    assert(contains(x));
    for (int i=0; i < size; i = i+1) {
      if (values[i].equals(x)) {
        values[i] = values[size-1];
        size = size - 1;
        break;
  } } }

  @Override
  public boolean contains(X x) {
    boolean contains = false;
    for (X value : values) {
      if (value.equals(x)) {
        contains = true;
        break;
    } }
    return contains; }

  @Override
  public int size() {
    return size; }

  @Override
  public void empty() {
    size = 0;
} }
```

Can we make our ArraySet iterable?

Iterable promises to provide an Iterator

```
interface Iterable<E> {
  public Iterator<E> iterator();
...} // shipped with Java
```

```
interface Iterator<E> {
  public boolean hasNext();
  public E next();
...} // shipped with Java
```

```
interface Set<X> {
  public void insert(X x);
  public void delete(X x);
  public void empty();
  public boolean contains(X x);
  public int size();
}
```

an Iterator provides all methods needed to step through all elements of a collection

```
class IteratorWorld {
  public static void main (
    String[] args) {
    int sum = 0;
    ...
    IterableArraySet<Integer> set =
      new IterableArraySet<>();
    set.insert(1);
    set.insert(2);
    ...
    for (Integer i : set) {
      sum += i.intValue();
      System.out.println(i);
    }
    System.out.println(sum);
} }
```

simple for-loop to iterate through the set

interface to implement

extending the ArraySet to add functionality to iterate over

```
import java.lang.Iterable;
import java.util.Iterator;

class IterableArraySet<X>
      extends ArraySet<X>
      implements
      Iterable<X> {
  @Override
  public Iterator<X> iterator() {
    return new Iterator<X>() {
      private int index = 0;
      public X next () {
        X x = values[index];
        index = index + 1;
        return x;
      }
      public boolean hasNext() {
        return (index < size);
  } };}
}
```

Iterator defined as anonymous class

The **Iterator** Pattern *in Detail*

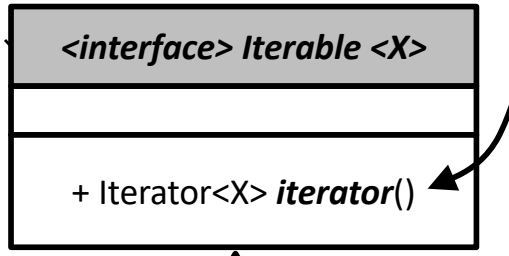
SYNOPSIS | UML | code | comments

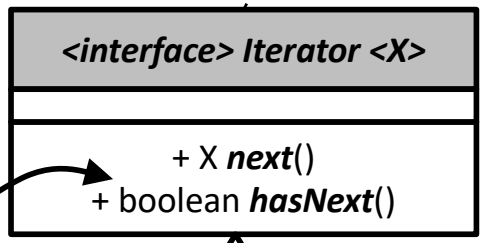
```
interface Iterable<X> {
  public Iterator<X> iterator();
} // shipped with Java
```

collection becomes 'Iterable' once an Iterator for it can be retrieved

```
interface Iterator<X> {
  public boolean hasNext();
  public X next();
} // shipped with Java
```

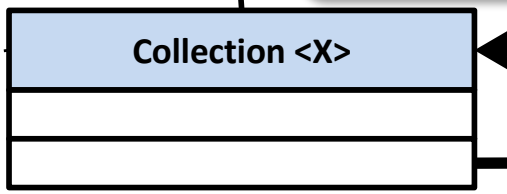
```
interface Set<X> {
  public void insert(X x);
  public void delete(X x);
  public void empty();
  public boolean contains(X x);
  public int size();
}
```

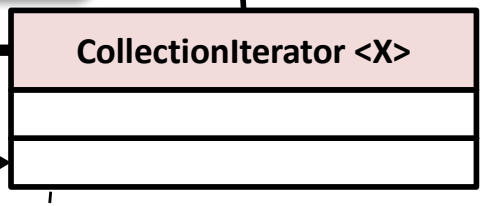





```
class ArraySet<X>
  implements Set<X> {
  protected X[] values;
  protected int size;
  ...
}
```

Iterator has to know how to step through all elements






```
class IterableArraySet<X>
        extends ArraySet<X>
        implements
        Iterable<X> {
  @Override
  public Iterator<X> iterator() {
    return new Iterator<X>() {
      private int index = 0;
      public X next () {
        X x = values[index];
        index = index + 1;
        return x;
      }
      public boolean hasNext() {
        return (index < size);
  } };}
}
```

The Iterator pattern is used to provide a standard interface for traversing a collection of items in an aggregate object without the need to understand its underlying structure. [GoF]

```
class IteratorWorld {
    ...
    IterableArraySet<Integer> set =
      new IterableArraySet<>();
    set.insert(1);
    set.insert(2);
    sum = 0;
    for (Integer i : set) {
      sum += i.intValue();
      System.out.println(i);
    }
    System.out.println(sum);
} }
```