CS2030S

Programming Methodology II

Lecture 07: Immutable and Nested Classes

Immutable Class

Immutable Class

Motivation

Classes

Point

Point_v0.java

```
class Point {
  private double x;
  private double y;

  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public void moveTo(double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

Circle

Circle_v0.java

```
class Circle {
  private Point c;
  private double r;

  public Circle(Point c, double r) {
    this.c = c;
    this.r = r;
  }

  public void moveTo(double x, double y) {
    c.moveTo(x, y);
  }
}
```

Immutable Class

Motivation

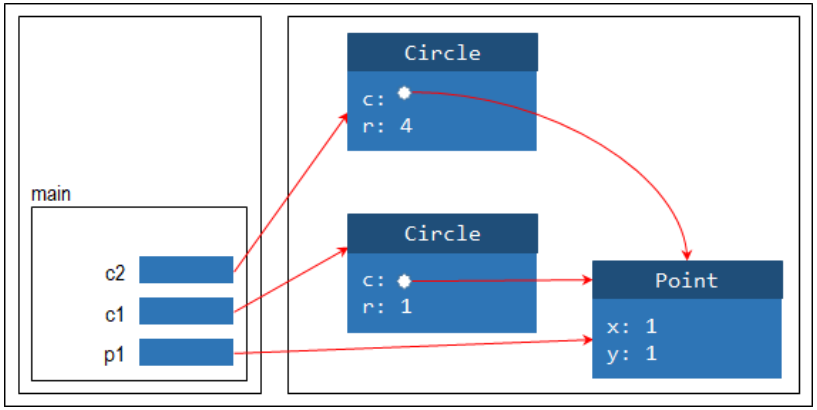
Moving Points

Aliasing

```
Point p1 = new Point(0, 0);
Circle c1 = new Circle(p1, 1);
Circle c2 = new Circle(p1, 4);
c1.moveTo(1, 1);
```

Question

Move circle `c1` *(and only `c1`)* to the point (1, 1).



Immutable Class

Definition

Immutable Class

A class is considered an **immutable class** if the instance of the class *cannot have any visible changes outside its abstraction barrier*.

Immutable Class

Towards Immutable Point

Attempt #0

Point

Point_v0.java

```
class Point {
  private double x;
  private double y;

  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public void moveTo(double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

Problem

```
jshell> Point p = new Point(0, 0)
p ==> (0.0, 0.0)
jshell> p
p ==> (0.0, 0.0)
jshell> p.moveTo(1, 1)
jshell> p
p ==> (1.0, 1.0)
```

Immutable Class

Towards Immutable Point

Attempt #1

Point

Point_v1.java

```
class Point {
  private final double x;
  private final double y;

  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public void moveTo(double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

Problem

```
jshell> /open Point.java
|  Error:
|  cannot assign a value to final ...
|      this.x = x;
|      ^----^
|  Error:
|  cannot assign a value to final ...
|      this.y = y;
|      ^----^
```

Immutable Class

Towards Immutable Point

Attempt #2

Point

Point_v2.java

```
class Point {
  private final double x;
  private final double y;

  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public Point moveTo(double x, double y) {
    Point pt = new Point(x, y);
    return pt;
  }
}
```

Problem ?

```
jshell> /open Point.java
jshell> Point p = new Point(0, 0)
p ==> (0.0, 0.0)
jshell> p.moveTo(1, 1)
$7 ==> (1.0, 1.0)
jshell> p
p ==> (0.0, 0.0)
```

Immutable Class

Towards Immutable Point

Attempt #2

Point

Point_v2.java

```
class Point {
  private final double x;
  private final double y;

  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public Point moveTo(double x, double y) {
    Point pt = new Point(x, y);
    return pt;
  }
}
```

Problem ?

```
class Point3D extends Point {
  private double x;  // bad practice
  private double y;  // bad practice
  private double z;

  @Override
  public Point moveTo(double x, double y) {
    this.x = x;
    this.y = y;
    return null;
  } // also override toString
}
```

Immutable Class

Towards Immutable Point

Attempt #3

Point

Point_v3.java

```
final class Point {
  private final double x;
  private final double y;

  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public Point moveTo(double x, double y) {
    Point pt = new Point(x, y);
    return pt;
  }
}
```

Question

What about `Circle`?

Immutable Class

Towards Immutable Circle

Attempt #0

Circle

Circle_v0.java

```
class Circle {
  private Point c;
  private double r;

  public Circle(Point c, double r) {
    this.c = c;
    this.r = r;
  }

  public void moveTo(double x, double y) {
    c.moveTo(x, y);
  }
}
```

Problem

```
jshell> Point p1 = new Point(0, 0)
p1 ==> (0.0, 0.0)
jshell> Circle c1 = new Circle(p1, 1)
c1 ==> (0.0, 0.0): 1.0
jshell> Circle c2 = new Circle(p1, 4)
c2 ==> (0.0, 0.0): 4.0
jshell> c1.moveTo(1, 1)
jshell> c1
c1 ==> (0.0, 0.0): 1.0
```

Immutable Class

Towards Immutable Circle

Attempt #1

Circle

Circle_v1.java

```
final class Circle {
  private final Point c;
  private final double r;

  public Circle(Point c, double r) {
    this.c = c;
    this.r = r;
  }

  public Circle moveTo(double x, double y) {
    return new Circle(c.moveTo(x, y), r);
  }
}
```

Back to Starting Point

```
jshell> /open Circle.java
jshell> Point p1 = new Point(0, 0)
p1 ==> (0.0, 0.0)
jshell> Circle c1 = new Circle(p1, 1)
c1 ==> (0.0, 0.0): 1.0
jshell> Circle c2 = new Circle(p1, 4)
c2 ==> (0.0, 0.0): 4.0
jshell> c1 = c1.moveTo(1, 1)
c1 ==> (1.0, 1.0): 1.0
jshell> c1
c1 ==> (1.0, 1.0): 1.0
jshell> c2
c2 ==> (0.0, 0.0): 4.0
```

Immutable Class

Condition

Immutable Class

A class is considered an **immutable class** if the instance of the class *cannot have any visible changes outside its abstraction barrier*.

Sufficiency

- Simply using the keyword `final` may not be sufficient
  - *e.g., it has a field with a type of mutable class and you have a getter.*

Necessity

- A class may still be immutable without the keyword `final`
  - *e.g., the class does not have any methods at all so how can it even be mutable?*

Question

Can you provide an example for each of the cases above?

Immutable Class

Advantages

Ease of Understanding

Example

```
Sensor s = new Sensor("detect child");
foo(s);  // may or may not turn off
bar(s);  // may or may not turn off
baz(s);  // may or may not turn off

// is sensor guaranteed to still be on?
if (s.detectChild()) {
  dontHitChildren();
}
```

Immutable Class

Advantages

Enable Safe Sharing of Internals

Motivation

> **Question**
>
> Create an immutable generic `ImmutableArray<T>` with two methods:
>
> 1. `T get(int index)`
>   - retrieves the element at the given index
> 2. `ImmutableArray<T> subarray(int start, int end)`
>   - returns a new `ImmutableArray<T>` containing only the sub-array

Notes

We will only discuss method 2 here.

Immutable Class

Advantages

Enable Safe Sharing of Internals

Naïve Approach

ImmutableArray_v0.java

```
class ImmutableArray<T> {
  private final T[] array;
  private final int start;
  private final int end;

  @SafeVarargs
  public static <T> ImmutableArray<T> of(T... items) {
    return new ImmutableArray<T>(items, 0, items.length);
  }

  public ImmutableArray<T> subarray(int start, int end) {
    // Create new T[] arr
    // Copy the content from this.array to arr
    return new ImmutableArray<T>(arr);
  }
}
```

Notes

- "`of(T... items)`" accepts a variable number of arguments of the same type `T`
  - This is called *variadic* function
  - You can pass 0 or more arguments into `of`
  - `items` is of type `T[]`

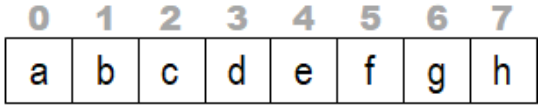
> **Question**
>
> Can we do it without copying the array?

Immutable Class

Advantages

Enable Safe Sharing of Internals

Concept

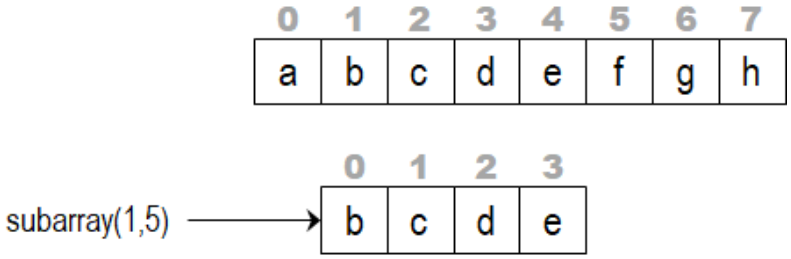


Immutable Class

Advantages

Enable Safe Sharing of Internals

Concept

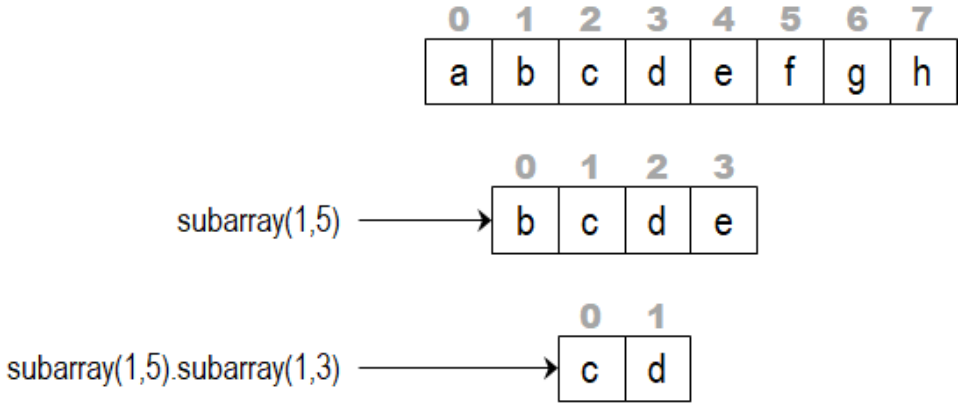


Immutable Class

Advantages

Enable Safe Sharing of Internals

Concept



Immutable Class

Advantages

Enable Safe Sharing of Internals

Sharing Approach

ImmutableArray_v1.java

```
class ImmutableArray<T> {
  private final T[] array;
  private final int start;
  private final int end;

  @SafeVarargs
  public static <T> ImmutableArray<T> of(T... items) {
    return new ImmutableArray<T>(items);
  }

  public ImmutableArray<T> subarray(int start, int end) {
    int newStart = this.start + start;
    int newEnd = this.start + end;
    return new ImmutableArray<T>(this.array, newStart, newEnd);
  }
}
```

Immutable Class

Advantages

Enable Safe Sharing of Objects

Empty Box

> Add a class method in `Box` called `empty()` that creates and returns an empty box, i.e., a box with a `null` item stored in it.
>
> Since empty boxes are likely common, we want to *cache* and reuse the empty box, that is, create one as a private final class field called `EMPTY_BOX`, and *whenever we need to return an empty box,* `EMPTY_BOX` *is returned*.

Immutable Class

Advantages

Enable Safe Concurrent Execution

Future Topic

Nested Class

Immutable Class

Class

Encapsulation

- Combine data that belongs together *(x- and y-coordinate of a point)*
- Combine methods that works on the data *(compute distance)*

Information Hiding

- Limits access to data
- Combined with "Tell, Don't Ask" principle

Problems

What if our class is not fine-grained enough?

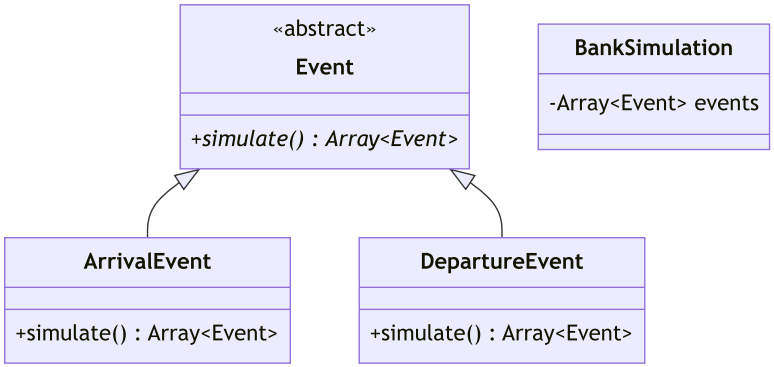
Immutable Class

Class
- Motivation
Kinds

Class

Motivation

Lab 1



Possible Design

Subclasses of `Event` *(e.g., `ArrivalEvent`, `DepartureEvent`, etc)* are not used outside `BankSimulation`.

Why not just put it as a class inside `BankSimulation`?

Immutable Class

Kinds of Nested Classes

Classification

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static

2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static

3. Local classes
   - Inside another class
   - Inside a method

4. Anonymous classes
   - Has no name

Immutable Class

Kinds of Nested Classes

Inner Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static

2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static

3. Local classes
   - Inside another class
   - Inside a method

4. Anonymous classes
   - Has no name

Immutable Class

Kinds of Nested Classes

Inner Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Example

```
class A {
  private int x;
  static int y;

  class B {
    void foo() {
      x = 1;
      y = 1;
    }
  } // Qns: Which of the above is OK?
}
```

	Choice	Comment	
A	`x = 1;`	*YES: non-static to non-static*	✅
B	`y = 1;`	*YES: non-static to static*	✅

Immutable Class

Kinds of Nested Classes

Inner Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Example

```
class A {
  private int x = 0;   // (1)
  static int y;

  class B {
    private int x = 1; // (2)
    void foo() {
      x = 2;
    }
  } // Qns: Which x is modified?
}
```

	Choice	Comment	
A	(1)	*NO*	✗
B	(2)	*YES*	✓

Immutable Class

Kinds of Nested Classes

Inner Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Example

```
class A {
  private int x = 0;   // (1)
  static int y;

  class B {
    private int x = 1; // (2)
    void foo() {
      A.this.x = 2;
    }
  } // Qns: Which x is modified?
}
```

	Choice	Comment	
A	(1)	*YES*	✓
B	(2)	*NO*	✗

Immutable Class

Kinds of Nested Classes

Static Nested Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Immutable Class

Kinds of Nested Classes

Static Nested Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Example

```
class A {
  private int x;
  static int y;

  static class B { // static
    void foo() {
      x = 1;
      y = 1;
    }
  } // Qns: Which of the above is OK?
}
```

	Choice	Comment	
A	`x = 1`	*NO: static to non-static*	✗
B	`y = 1`	*YES: static to static*	✓

Immutable Class

Class
Kinds
- Classification
- Inner Classes
- Static Nested
- Local Nested
- Anonymous Classes

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Preliminary (Comparator)

Java SE 17 & JDK 17

Module java.base
Package java.util

Interface Comparator<T>

Type Parameters:

T - the type of objects that may be compared by this comparator

All Known Implementing Classes:

`Collator`, `RuleBasedCollator`

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

```
@FunctionalInterface
```

Immutable Class

Class
Kinds
- Classification
- Inner Classes
- Static Nested
- Local Nested
- Anonymous Classes

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Preliminary (List<T>)

Java SE 17 & JDK 17

Module java.base
Package java.util

Interface List<E>

Type Parameters:

E - the type of elements in this list

All Superinterfaces:

`Collection`<E>, `Iterable`<E>

All Known Implementing Classes:

`AbstractList`, `AbstractSequentialList`, `ArrayList`, `AttributeList`, `CopyOnWriteArrayList`, `LinkedList`, `RoleList`, `RoleUnresolvedList`, `Stack`, `Vector`

Immutable Class

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Motivation

> **Question**
>
> Write a method to sort a list of names *(of type `String`)* based on their length only.

Immutable Class

Class
Kinds
- *Classification*
- *Inner Classes*
- *Static Nested*
- ***Local Nested***
- *Anonymous Classes*

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Problem

```
interface C { void g(); }
class A {
  int x = 1;
  C f() {
    int y = 2;
    class B implements C {
      public void g() {
        x = y; // accessing x and y is OK.
      }
    }
    B b = new B();
    return b;
  }
}
```

```
A a = new A();
C c = a.f();
c.g();
```

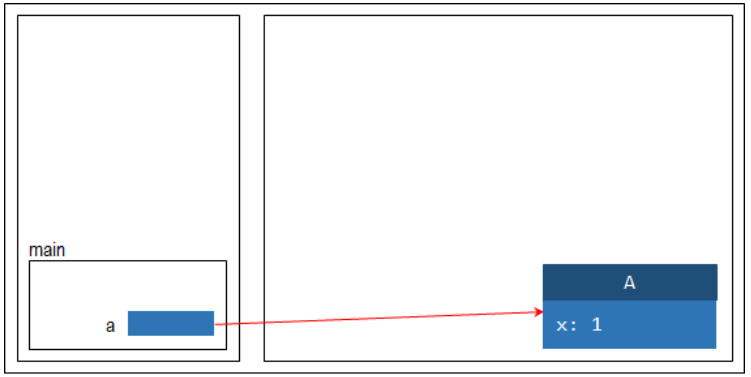
Immutable Class

Class
Kinds
- *Classification*
- *Inner Classes*
- *Static Nested*
- ***Local Nested***
- *Anonymous Classes*

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name



Problem

```
interface C { void g(); }
class A {
  int x = 1;
  C f() {
    int y = 2;
    class B implements C {
      public void g() {
        x = y; // accessing x and y is OK.
      }
    }
    B b = new B();
    return b;
  }
}
```

```
A a = new A();
C c = a.f();
c.g();
```

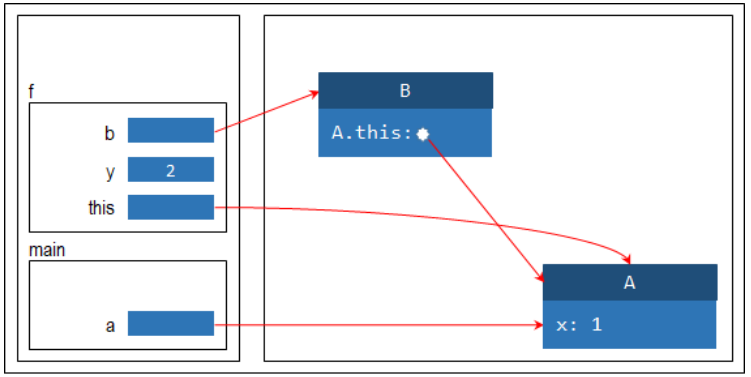
Immutable Class

Class
Kinds
- *Classification*
- *Inner Classes*
- *Static Nested*
- ***Local Nested***
- *Anonymous Classes*

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name



Problem

```
interface C { void g(); }
class A {
  int x = 1;
  C f() {
    int y = 2;
    class B implements C {
      public void g() {
        x = y; // accessing x and y is OK.
      }
    }
    B b = new B();
    return b;
  }
}
```

```
A a = new A();
C c = a.f();
c.g();
```

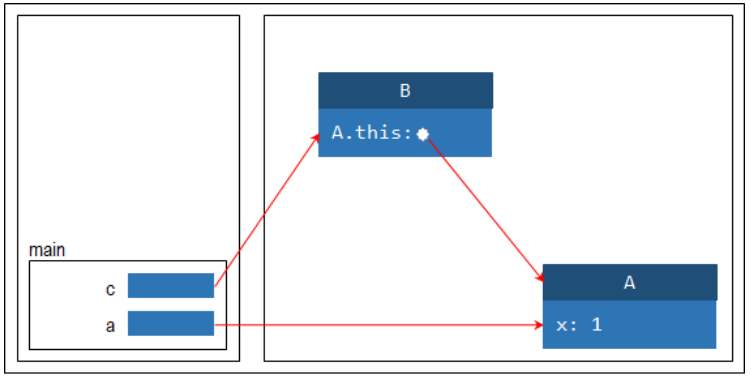
Immutable Class

Class
Kinds
- *Classification*
- *Inner Classes*
- *Static Nested*
- ***Local Nested***
- *Anonymous Classes*

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name



Problem

```
interface C { void g(); }
class A {
  int x = 1;
  C f() {
    int y = 2;
    class B implements C {
      public void g() {
        x = y; // where is y??
      }
    }
    B b = new B();
    return b;
  }
}
```

```
A a = new A();
C c = a.f();
c.g();
```

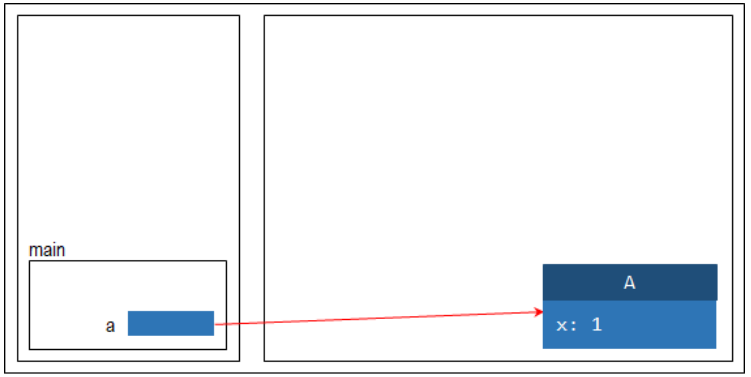
Immutable Class

Class
Kinds
- *Classification*
- *Inner Classes*
- *Static Nested*
- ***Local Nested***
- *Anonymous Classes*

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name



Variable Capture

```
interface C { void g(); }
class A {
  int x = 1;
  C f() {
    int y = 2;
    class B implements C {
      public void g() {
        x = y; // accessing x and y is OK.
      }
    }
    B b = new B();
    return b;
  }
}
```

```
A a = new A();
C c = a.f();
c.g();
```

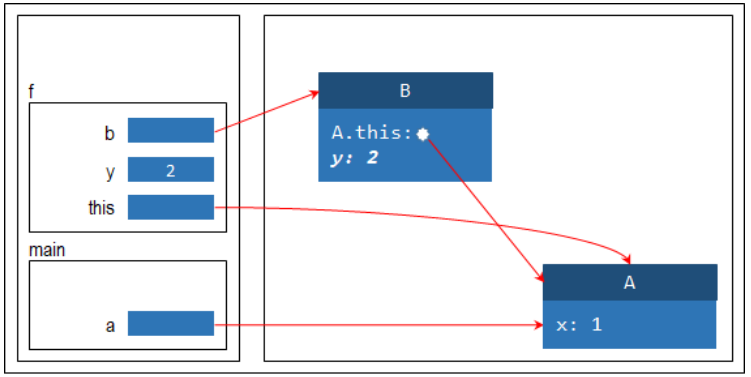
Immutable Class

Class
Kinds
- *Classification*
- *Inner Classes*
- *Static Nested*
- ***Local Nested***
- *Anonymous Classes*

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name



Variable Capture

```
interface C { void g(); }
class A {
  int x = 1;
  C f() {
    int y = 2;
    class B implements C {
      public void g() {
        x = y; // accessing x and y is OK.
      }
    }
    B b = new B();
    return b;
  }
}
```

```
A a = new A();
C c = a.f();
c.g();
```

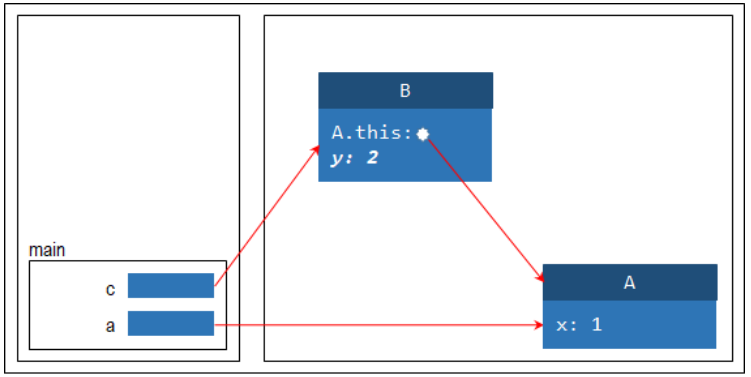
Immutable Class

Class
Kinds
- *Classification*
- *Inner Classes*
- *Static Nested*
- ***Local Nested***
- *Anonymous Classes*

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name



Variable Capture

```
interface C { void g(); }
class A {
  int x = 1;
  C f() {
    int y = 2;
    class B implements C {
      public void g() {
        x = y; // can we assign to y? (e.g., y = 3)
      }
    }
    B b = new B();
    return b;
  }
}
```

```
A a = new A();
C c = a.f();
c.g();
```

Immutable Class

Class
Kinds
- *Classification*
- *Inner Classes*
- *Static Nested*
- ***Local Nested***
- *Anonymous Classes*

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Effectively Final

```
void sortNames(List<String> names) {
  boolean ascendingOrder = true;
  class NameComparator implements Comparator<String> {
    public int compare(String s1, String s2) {
      if (ascendingOrder) { // is it true or false?
        return s1.length() - s2.length();
      } else {
        return s2.length() - s1.length();
      }
    }
  }

  ascendingOrder = false;
  names.sort(new NameComparator());
}
```

Immutable Class

Kinds of Nested Classes

Local Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

Effectively Final (Equivalently)

```
void sortNames(List<String> names) {
  final boolean ascendingOrder = true;
  class NameComparator implements Comparator<String> {
    public int compare(String s1, String s2) {
      if (ascendingOrder) { // is it true or false?
        return s1.length() - s2.length();
      } else {
        return s2.length() - s1.length();
      }
    }
  }

  // ascendingOrder = false; <-- cannot change final
  names.sort(new NameComparator());
}
```

Immutable Class

Kinds of Nested Classes

Anonymous Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

From Named to Nameless

```
void sortNames(List<String> names) {

  class NameComparator implements Comparator<String> {
    @Override
    public int compare(String s1, String s2) {
      return s1.length() - s2.length();
    }
  }

  names.sort(new NameComparator());
}
```

Immutable Class

Kinds of Nested Classes

Anonymous Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

From Named to Nameless

```
void sortNames(List<String> names) {

  Comparator<String> cmp = Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
      return s1.length() - s2.length();
    }
  };

  names.sort(cmp);
}
```

Immutable Class

Kinds of Nested Classes

Anonymous Classes

1. Inner classes
   - Inside another class
   - Not inside a method
   - Not static
2. Static nested classes
   - Inside another class
   - Not inside a method
   - Static
3. Local classes
   - Inside another class
   - Inside a method
4. Anonymous classes
   - Has no name

From Named to Nameless

```
void sortNames(List<String> names) {

  names.sort(Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
      return s1.length() - s2.length();
    }
  });

  // names.sort(cmp); <-- no need anymore
}
```

```
jshell> /exit
|    Goodbye
```