

# INNER CLASSES

# Why?

- Promote the logical grouping of related classes
- Increase encapsulation and information hiding
- Results in more analyzable and maintainable code.

# Inner classes

Inner (or nested) classes are classes defined within other classes:

inner classes in Java:

- **Member classes**

- Simple and useful

- **Local (including Anonymous) classes**

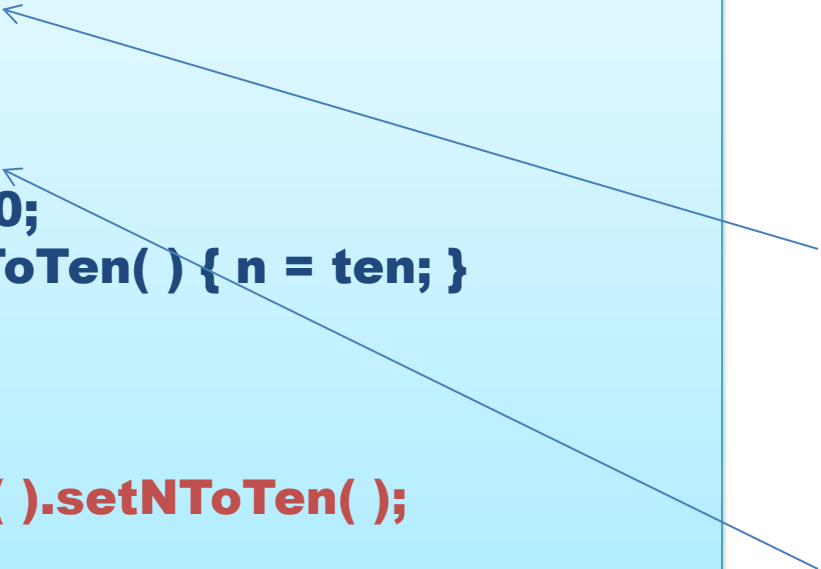
- Useful, but syntax is ugly

# Inner and Outer Classes Have Access to Each Other's Private Members

- Within the definition of a method of **an inner class**:
  - It is legal to reference a **private** instance variable of the outer class
  - It is legal to invoke a **private** method of the outer class
- Within the definition of a method of **the outer class**
  - It is legal to reference a **private** instance variable of the inner class on an object of the inner class
  - It is legal to invoke a method of the inner class as long as an object of the inner class is used as a calling object
- Within the definition of the inner or outer classes, the modifiers **public** and **private** are equivalent

# Member classes

Often used to describe part\_of relationships



```
class Outer {  
  int n;  
  
  class Inner {  
    int ten = 10;  
    void setNToTen( ) { n = ten; }  
  }  
  
  void setN ( ) {  
    new Inner( ).setNToTen( );  
  }  
}
```

```
public class Outer
```

```
    private int counter=0;
```

```
    public class Inner{  
        public void someMethod(){  
            counter++;  
        }  
    }
```

```
    public int getCount(){  
        return counter;  
    }
```

```
}
```

```
public class InnerClassExample

    public static void main (String []) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();
        inner.someMethod();
        System.out.println( outer.getCount());
        inner.someMethod();
        System.out.println( outer.getCount());
    }
}
```

# More meaningful example

- Generate a random list of odd integers
- Algorithm
  - Generate a fixed number of integers
  - Iterate through and discard even numbers

Reader be aware the following slides has some  
super evil code!



```
public class DynamicOddsGenerator

private final static int SIZE = 25;
private int[] arrayOfInts = new int[SIZE];

public DynamicOddsGenerator(){
    for (int i =0; i < SIZE; i++) {
        arrayOfInts[i] = (int) (Math.random()*SIZE);
    }
}
```

```
public void printOdds(){  
  
    InnerOddsIterator iterator = this.new InnerOddsIterator();  
  
    while (iterator.hasNext()){  
        int returnValue = iterator.getNext();  
        if (returnValue != -1{  
            System.out.print(returnValue + " ");  
        }  
    }  
    System.out.println();  
}
```

```
private class InnerOddsIterator {  
  
    private int next = 0;  
  
    public boolean hasNext(){  
        return (next <= SIZE -1);  
    }  
    public int getNext(){  
        int returnValue = arrayOfInts[next++]  
        if (returnValue % 2 == 1) {  
            return returnValue;  
        }  
        return -1;  
    }  
}
```

```
// back in the outer class
```

```
public static void main(String s[]) {
```

```
    DynamicOddsGenerator numbers =  
        new DynamicOddsGenerator();
```

```
    numbers.printOdds();
```

```
}
```

```
} // end outer class
```

# Local class

- A local class is defined within a block of Java code.
- Local classes are completely hidden in their containing block.
- When a class name is used only within a block it can be defined locally.
- A local class can access instance variables of the outer class and only the **final** local variables of the enclosing block.

```
class LocalClassExample{
    private String name = "UofA";
    public void method ( ) {
        int j = 20;
        final int k = 30;
        class Local {
            public void test ( ) {
                //System.out.println(j); //Error as j is not final
                System.out.println(k); //OK k is final

                //Like an inner class, instance variables of
                //the enclosing object can be accessed.
                System.out.println ( name ) ;
            }
        }
        Local loc = new Local ( ) ;
        loc.test ( ) ;
    }

    public static void main ( String [ ] args ) {
        LocalClassExample obj = new LocalClassExample ( );
        obj.method ( ) ;
    }
}
```

# The `.class` File for an Inner Class

- Compiling any class in Java produces a `.class` file named `ClassName.class`
- Compiling a class with one (or more) inner classes causes both (or more) classes to be compiled, and produces two (or more) `.class` files
  - Such as `ClassName.class` and `ClassName$InnerClassName.class`
- *Every* class compiles to a separate `.class` file
- Inner classes compile to files with a `$` in their names

# Anonymous class

- If only one object has to be created from a class, and there is no need to name the class, then an **anonymous class** definition can be used
  - The class definition is embedded inside the expression with the **new** operator
- Anonymous class has no constructors
- It is either derived from a class, or implements an interface. Like:
  - `AnInterface i = new AnInterface ( ) { // methods defs. ... }`
  - `ASuperclass c = new ASuperclass(...) { // methods defs. ... }`



# Anonymous Classes

## Display 13.11 Anonymous Classes (Part 1 of 2)

---

```
1 public class AnonymousClassDemo
2 {
3     public static void main(String[] args)
4     {
5         NumberCarrier anObject =
6             new NumberCarrier()
7             {
8                 private int number;
9                 public void setNumber(int value)
10                {
11                    number = value;
12                }
13                public int getNumber()
14                {
15                    return number;
16                }
17            };

```

*This is just a toy example to demonstrate the Java syntax for anonymous classes.*

# Anonymous Classes

## Display 13.11 Anonymous Classes (Part 1 of 2)

---

```
18         NumberCarrier anotherObject =
19             new NumberCarrier()
20             {
21                 private int number;
22                 public void setNumber(int value)
23                 {
24                     number = 2*value;
25                 }
26                 public int getNumber()
27                 {
28                     return number;
29                 }
30             };

31         anObject.setNumber(42);
32         anotherObject.setNumber(42);
33         showNumber(anObject);
34         showNumber(anotherObject);
35         System.out.println("End of program.");
36     }

37     public static void showNumber(NumberCarrier o)
38     {
39         System.out.println(o.getNumber());
40     }

41 }
```

*This is still the file  
AnonymousClassDemo.java.*

---

# Anonymous Classes

## Display 13.11 Anonymous Classes (Part 2 of 2)

---

### SAMPLE DIALOGUE

42  
84  
End of program.

```
1 public interface NumberCarrier
2 {
3     public void setNumber(int value);
4     public int getNumber();
5 }
```

*This is the file  
NumberCarrier.java.*

---

Why on earth would I do this?

# An anonymous inner class

- Anonymous classes in java are defined in a different way than that of a normal java classes.
- In Java anonymous classes can be created in either of the two ways.
  - 1) Using a class reference variable.
  - 2) Using an interface.

# Creating a anonymous inner class using a class's reference variable

```
public class Apple {  
    public void print() {  
        System.out.println("Printed from Apple !");  
    }  
  
    public void printAgain() {  
        System.out.println("Printed from Apple again !");  
    }  
}
```

**public class** Implementation {

/\* This looks like we are creating an object of Apple class, but we have created an instance of anonymous "subclass" of Apple.

The statement creates an anonymous inner class with an instance of it and the class is a "subclass" to Apple. \*/

/\* Note: we have thrown away all code (such as the surrounding method call) to simplify the presentation \*/

```
Apple apple = new Apple() {  
    @Override public void print() {  
        System.out.println("Printed from  
                               subclass of Apple !");  
    }  
}; // end Apple “subclass”  
} //end implementation
```

An anonymous inner class!



# Anonymous inner class in java is a “subclass”

- We are creating an object of Subclass with a reference variable of Superclass(Apple).
- Now this is the thing where Polymorphism comes into picture, from Subclass-Superclass Relationship we know that using ‘apple’ we can access members that are present in the Apple (“Superclass”).

```
public class Apple {  
    public void print() {  
        System.out.println  
            ("Printed from Apple !");  
    }  
}
```

```
public class Implementation {  
    Apple apple = new Apple() {  
        @Override public void print() {  
            System.out.println  
                ("Printed from subclass of Apple !");  
        }  
    }  
}
```

```
/* Not a Overridden method. */
```

```
public void add() {
```

```
    System.out.println("I am not in Apple !"); }
```

```
}; //end Apple subclass
```

```
public void r() {
```

```
    apple.print(); /* This will work fine. */
```

```
    apple.add(); /* compilation error */
```

```
/*because, from a superclass reference variable  
we can not call a member of subclass that is not  
present in superclass. */ }
```

# Creating an anonymous inner class from an interface

We can create an anonymous inner class that implements `AppleInterface` and an instance of this class at the same time.

This is the only condition in java, where we can implement an interface without even having a name of the class and without using the `implement` keyword.

```
public interface AppleInterface {  
    public void callMe(); }  

```

```
public class Implementation {  
    AppleInterface appleInterface = new AppleInterf  
ace() {  

```

```
@Override public void callMe() {  
    /* Write body! I need an implementation */  
} };
```

# Why to use an anonymous inner class in java

- Till now we have seen that, we have created a class 'Apple'; and than an anonymous inner class to override a method 'print()'.
- This thing can be done by creating a subclass of 'Apple' using 'extends' keyword, than why do we need to create a anonymous class?

# Why to use an anonymous inner class in java

- The answer to the question is, creating anonymous class is quicker and simple.
- Anonymous inner classes are useful when we need to inherit a **few** properties (**only one method commonly**) of a superclass.
- This is not a good idea to take overhead of creating a separate subclass for doing things so simple.



# Anonymous classes in Java

- Anonymous classes are just like local classes, besides they don't have a name. In Java anonymous classes enables the developer to declare and instantiate a class at the same time.
- In java, normal classes are declarations, but anonymous classes are expressions, so anonymous classes are created in expressions.
- Like other inner classes, an anonymous class has access to the members of its enclosing class.

# Consider the following definitions

## **Interface Comparator<T>**

```
int compare(T o1,T o2)
```

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

# Consider the following definitions

## Class Arrays

various methods for manipulating arrays

```
public static <T> void sort(T[] a, Comparator<? super T> c)
```

Sorts the specified array of objects according to the order specified by comparator. All elements in the array must be *mutually comparable* by the specified comparator (that is, `c.compare(e1, e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the array).

```
public class StupidComparator {  
    public static void main(String[] args) {  
  
        final int numberToCompareTo=10;  
  
        Comparator<Integer> comp=new  
            Comparator<Integer>() {
```

```
public int compare(Integer a, Integer b) {  
    int result=0;  
    if (a<numberToCompareTo)  
        result=result-1;  
    if (b<numberToCompareTo)  
        result=result+1;  
    return result;  
}  
}; // close comp
```

Truly ugly code –  
how would I fix it?

```
Integer[] array=new Integer[] {1,10, 5 , 15, 6 ,  
20, 21, 3, 7};
```

```
// this is a function call that takes the inner  
class comp as a parameter
```

```
Arrays.sort(array,comp);
```

```
for (int i:array) System.out.println(i);
```

```
}
```

```
}
```

# One more time .....

```
class Book{
    String title;
    int pageNumber;

    public Book(String title, int pageNumber){
        this.title = title;
        this.pageNumber = pageNumber;    }

    String getTitle(){ return title; }
    int getPageNumber(){ return pageNumber; }

    @override public String toString(){
        return "(" + title + ", " + pageNumber + " pages)"; }
}
```

```
public class Library{
```

```
    private Comparator<Book> ascTitle;
```

```
    private Comparator<Book> descPageNumber;
```



```
ascTitle = new Comparator<Book>(){  
    @Override  
    public int compare(Book b1, Book b2){  
        return b1.getTitle().compareTo(b2.getTitle());  
    }  
};
```

```
descPageNumber = new Comparator<Book>(){  
    @Override  
    public int compare(Book b1, Book b2){  
        return b2.getPageNumber() - b1.getPageNumber();  
    }  
};
```

```
private Book[] books;  
    public Book[] getBooks(){ return books; }  
  
    public void sortAscTitle(){  
        Arrays.sort(books, ascTitle);  
    }  
  
    public void sortDescPageNumber(){  
        Arrays.sort(books, descPageNumber);  
    }
```

```
public Library(Book[] books){  
    this.books = books;  
}
```

```
public static void main(String[] args){  
    Library library = new Library( new Book[]{  
        new Book("1984", 123),  
        new Book("I, Robot", 152),  
        new Book("Harry Potter and the Philosopher's Stone", 267),  
        new Book("Harry Potter and the Goblet of Fire", 759),  
        new Book("The Bible", 1623)  
    });
```

```
library.sortAscTitle();  
System.out.println(Arrays.toString(library.getBooks()));
```

```
library.sortDescPageNumber();  
System.out.println(Arrays.toString(library.getBooks()));
```

```
}
```

```
}
```