Nested Clases & Lambda Expressions

Java 21

2025

Objectives

- After completing this lesson, you should be able to use:
  - Nested classes
    - Static
    - Member
    - Local
    - Anonymous
  - Lambda expressions

Types of Nested Classes

Classes can be defined inside other classes to encapsulate logic and constrain context of use.

- Type of the nested class depends on the context in which it is used.
  - Static nested class is associated with the static context of the outer class.
  - Member inner class is associated with the instance context of the outer class.
  - Local inner class is associated with the context of a specific method.
  - Anonymous inner class is an inline implementation or extension of an interface or a class.
- Static and member nested classes can be defined as:
  - `public`, `protected`, or `default` - can be accessed externally
  - `private` - can be referenced only inside their outer class

```
public class Outer {
  public static class StaticNested {
    // code of the nested class
  }
}
Outer.StaticNested x = new Outer.StaticNested();
```

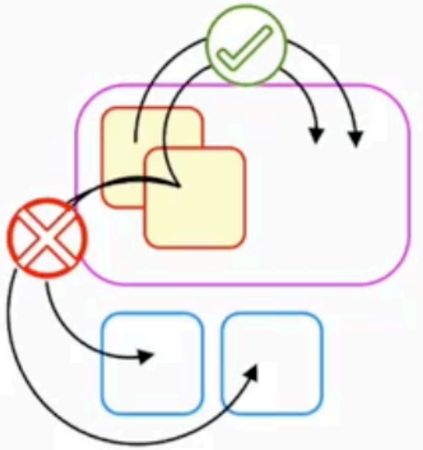
```
public class Outer {
  public static void createInstance() {
    new StaticNested();
  }
  private static class StaticNested {
    // code of the nested class
  }
}
Outer.createInstance();
```

Static Nested Classes

Static nested class is associated with the static context of the outer class.

- To create an instance of a static nested class, you do not need to create instances of outer class.
- Can access private variables and methods of the outer class
- Can only access static variables and methods of the outer class

```
public class Order {
  public static void createShippingMode(String description) {
    new ShippingMode(description);
  }
  private static class ShippingMode {
    private String description;
    public ShippingMode(String description) {
      this.description = description;
    }
    // other methods and variables of the ShippingMode class
  }
}
```



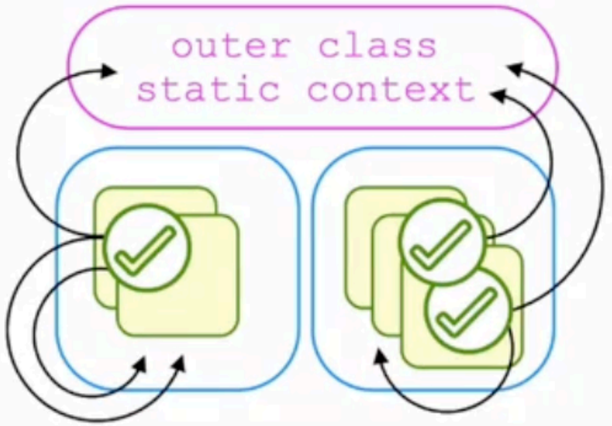
```
Order.createShippingMode("Fast");
Order.createShippingMode("Normal");
Order order1 = new Order();
Order order2 = new Order();
```

Member Inner Classes

Member inner class is associated with the instance context of the outer class.

- To create an instance of a member inner class, you must create an instance of outer class first.
- Can access private variables and methods of the outer class
- Can access both static and instance variables and methods of the outer class

```
public class Order {
  private Set<Item> items = new HashSet<>();
  public void addItem(Product product, int quantity) {
    items.add(new Item(product, quantity));
  }
  class Item {
    private Product product;
    private int quantity;
    private Item(Product product, int quantity) {
      this.product = product;
      this.quantity = quantity;
    }
    // other methods of the Item class
  }
}
```

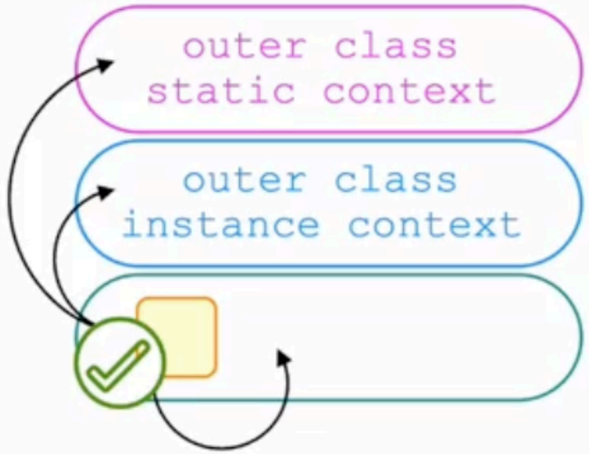



```
Order order1 = new Order();
Order order2 = new Order();
order1.addItem(new Drink("Tea"), 2);
order1.addItem(new Food("Cake"), 1);
order2.addItem(new Drink("Tea"), 1);
```

Local Inner Classes

Local inner class is associated with the context of a specific method.

- Instances of the local inner class can only be created within the outer method context.
- It contains logic complex enough to require the algorithms be wrapped up as a class.
- Outer method local variables and parameters can only be accessed if they are final or effectively final.




```
public class Order {
  private Map<Integer, Item> items = new HashMap<>();
  public void manageTax(final String saleLocation) {
    class OrderTaxManager {
     private BigDecimal findRate(Product product) {
       // use saleLocation and product to find the tax rate
     }
     BigDecimal calculateTax() {
       // find tax rate in a given sale location for each product
       // calculate tax value
     }
    }
    OrderTaxManager taxManager = new OrderTaxManager();
    BigDecimal taxTotal = taxManager.calculateTax();
  }
}
```

Anonymous Inner Classes

Anonymous inner class is an implementation of an interface or extension of a class.

- It extends a parent class or implement an interface to override operations.
- It is implemented inline and instantiated immediately.
- Outer method local variables and parameters can only be accessed if they are final or effectively final.

```
public class Order {
  public BigDecimal getDiscount() {
    return BigDecimal.ZERO;
  }
}
```

Separate Class Implementation

```
public class OnlineOrder extends Order {
  @Override
  public BigDecimal getDiscount() {
    return BigDecimal.valueOf(0.1);
  }
}
```

Anonymous Inner Class Implementation

```
Order order = new Order() {
  @Override
  public BigDecimal getDiscount() {
    return BigDecimal.valueOf(0.1);
  }
};
```

Anonymous Inner Classes and Functional Interfaces

Anonymous inner classes are typically used to provide inline interface implementations.

- Anonymous inner class can implement an interface inline and override as many methods as required.
- Functional interfaces define only one abstract method that must be overridden.
- Anonymous inner class that implements functional interface will only have to override one method.
- It could be more convenient to:
  - Use a regular class to override many methods
  - Use anonymous inner class to override a few methods (just one in case of a functional interface)

```
List<Product> products = ...
Collections.sort(products, new Comparator<Product>(){
  public int compare(Product p1, Product p2) {
    return p1.getName().compareTo(p2.getName()));
  }
});

Collections.sort(products, new Comparator<Product>(){
  public int compare(Product p1, Product p2) {
    return p1.getPrice().compareTo(p2.getPrice()));
  }
});
```

Lambdas expressions