**Inner, Anonymous and Lambda Workshop**

Java Inner Class | DigitalOcean

**01**

Start with this code skeleton:

class Car {

    private String brand;

}

class Main {

    public static void main(String [] args) {

        Car datsun = new Car();

    }

}

In the main method, try to assign the datsun object a brand called "Datsun 100a".

* Why doesn't it compile?
* What is the visibility of a private variable?

Instead implement a Motor inner class inside the Car class.

class Car {

    private String brand;

    class Motor {

    }

}

Compile the code. Notice that there are 3 class files created in the directory.

Car.class

Car$Motor.class

Main.class

Add the following method to the Motor class

public void printCarBrand() {

    System.out.println(brand);

}

Compile the code.

* Can the Motor inner class, use the Car outer class private variables?
* Did the compiler complain about private variables?

Next add a Motor attribute to the Car class.

class Car {

    private String brand;

    private Motor motor;

Next implement a constructor for the Car class that takes the the brand as parameter and but instantiate the Motor object.

public Car(String brand) {

    this.brand = brand;

    this.motor = new Motor();

}

Next implement a get method that returns the Motor object, add the method to the Car class.

public Motor getMotor() {

    return this.motor;

}

… in the main method, create a "Datsun 100a" brand car and print to the console the brand using the method of the Motor class.

light bulb Hint datsun.getMotor()....

// Expected output

Datsun 100a

**Solution**

class Car {

    private String brand;

    private Motor motor;

    public Car(String brand) {

        this.brand = brand;

        this.motor = new Motor();

    }

    public Motor getMotor() {

        return this.motor;

    }

    class Motor {

        public void printCarBrand() {

            System.out.println(brand);

        }

    }

}

class Main {

    public static void main(String [] args) {

        Car datsun = new Car("Datsun 1000a");

        datsun.getMotor().printCarBrand();

    }

}

**02**

Start with the code skeleton:

class Main {

    public static void main(String [] args) {

    }

}

Create a Bird class that only have a constructor that prints to screen that "Hello Birdie!".

class Bird {

    public Bird() {

        // "Hello Birdie!"

    }

}

Move the Bird class into the main method. Create a Bird object and notice the bird greeting.

What do you need to remember when adding the class to main? Sequence matters.

Change the constructor to print a variable named text.

class Bird {

    public Bird() {

        System.out.println(text);

    }

}

Now declare the text variable in the main method.

class Main {

    public static void main(String [] args) {

        String text = "Hello Birdie!";

        class Bird { .. }

    }

}

Inner classes can use the outer classes variables.

**Solution**

class Main {

    public static void main(String [] args) {

        final String text = "Hello Birdie!";

        class Bird {

            public Bird() {

                System.out.println(text);

            }

        }

        Bird bird = new Bird();

    }

}

**03**

Start with this code skeleton:

class Bird {

    public void fly() {

        System.out.println("Fly Birdie Fly!");

    }

}

class Main {

    public static void main(String [] args) {

    }

    public static void fly(Bird b) {

        b.fly();

    }

}

In the main method create a Bird object and call the fly method with the object as parameter. The program prints "Fly Birdie Fly!".

In the main method add an Ostrich inner class that inherits the Bird class.

class Ostrich extends Bird {

}

Replace the Ostrich fly method to display "Ostriches wish they can fly!". Remember to use the @Override annotation.

@Override annotation informs the compiler that the element is meant to override an element declared in a superclass.

Now create an Ostrich object in the main method and pass it as parameter to the fly method.

// Expected output

Fly Birdie Fly!

Ostriches wish they can fly!

**Solution**

class Bird {

    public void fly() {

        System.out.println("Fly Birdie Fly!");

    }

}

class Main {

    public static void main(String [] args) {

        Bird bird = new Bird();

        fly(bird);

        class Ostrich extends Bird {

            @Override

            public void fly() {

                System.out.println("Ostriches wish they can fly!");

            }

        }

        Ostrich ostrich = new Ostrich();

        fly(ostrich);

    }

    public static void fly(Bird b) {

        b.fly();

    }

}

**04**

Start with solution above from exercise 03.

The solution contains 3 classes at the moment:

Bird

Ostrich

Main

These three classes are all named classes. Next create an anonymous class, a class that is inside a method that has no name.

First declare a new Bird variable in the main method, Bird x;

Any Bird object or object that inherits from Bird can be assigned to the x variable.

Try the following code:

Bird x = new Bird() {

    @Override

    public void fly() {

        System.out.println("Look! An UFO!");

    }

};

x.fly();

The code above creates a anonymous class inside the main method that inherits from the Bird class.

class Anonymous extends Bird {

    @Override

    public void fly() {

        System.out.println("Look! An UFO!");

    }

}

It then creates an object from the anonymous class

Anonymous nameless = new Anonymous();

It assigns this nameless object to x.

Next try to pass the whole anonymous class as parameter to the fly method.

fly(new Bird() {});

// Expected Output

Fly Birdie Fly!

Ostriches wish they can fly!

Look! An UFO!

**Solution**

class Bird {

    public void fly() {

        System.out.println("Fly Birdie Fly!");

    }

}

class Main {

    public static void main(String [] args) {

        Bird bird = new Bird();

        fly(bird);

        class Ostrich extends Bird {

            @Override

            public void fly() {

                System.out.println("Ostriches wish they can fly!");

            }

        }

        Ostrich ostrich = new Ostrich();

        fly(ostrich);

        fly(new Bird() {

            @Override

            public void fly() {

                System.out.println("Look! An UFO!");

            }

        });

    }

    public static void fly(Bird b) {

        b.fly();

    }

}

**05**

Start with this code skeleton:

class Main {

    public static void main(String [] args) {

    }

    public static void sell(SalesItem item) {

        item.sell();

    }

}

Create an interface called SalesItem that has 1 method called sell.

In the main method create a class called EnergyDrink that implements the SalesItem interface.

The sell method should print "EnergyDrink sold". (Remember to use @Override).

Create an EnergyDrink object and pass it as parameter to the sell method.

EnergyDrink ed = new EnergyDrink();

sell(ed);

Create another class called Dog that implements the SalesItem interface. The sell method should print "Doggie sold!".

Dog spot = new Dog();

sell(spot);

// Expected output

EnergyDrink sold!

Doggie sold!

Now implement an anonymous class that implement the interface. In the main method declare a variable

SalesItem x;

Any object that implements the SalesItem interface can be assigned to the x variable.

Next implement the interface inside the anonymous class

SalesItem x = new SalesItem() {

    @Override

    public void sell() {

        System.out.println("Nameless item sold!");

    }

};

sell(x);

Compile and run the code.

// Expected output

EnergyDrink sold!

Doggie sold!

Nameless item sold!

Next try to pass the whole anonymous class as a parameter to the sell method.

sell(new SalesItem() { });

Compile and run the code.

// Expected output

EnergyDrink sold!

Doggie sold!

Nameless item sold!

**Solution**

interface SalesItem {

    void sell();

}

class Main {

    public static void main(String [] args) {

        class EnergyDrink implements SalesItem {

            @Override

            public void sell() {

                System.out.println("EnergyDrink sold!");

            }

        }

        EnergyDrink ed = new EnergyDrink();

        sell(ed);

        class Dog implements SalesItem {

            @Override

            public void sell() {

                System.out.println("Doggie sold!");

            }

        }

        Dog spot = new Dog();

        sell(spot);

        SalesItem x = new SalesItem() {

            @Override

            public void sell() {

                System.out.println("Nameless item sold!");

            }

        };

        sell(x);

        sell(new SalesItem() {

            @Override

            public void sell() {

                System.out.println("Another nameless item sold!");

            }

        });

    }

    public static void sell(SalesItem item) {

        item.sell();

    }

}

**06**

Start with solution above from exercise 05.

**If** an interface, like SalesItem has exactly **one** method **and** you are using **Java8 or above**, you can use lambda expressions.

Currently we pass the whole anonymous class as a parameter to the sell method.

sell(new SalesItem() {

    @Override

    public void sell() {

        System.out.println("Another nameless item sold!");

    }

});

Replacing it with a lambda:

sell( () -> System.out.println("lambda sold!"));

Compile and run.

In the Main class add a new method

public static void method() {

    System.out.println("Method reference sold");

}

Call the method using a lambda

sell(() -> method());

Compile and run, the text Method reference sold should have been printed to the console.

If the lambda is calling only one method, you can use a method reference.

sell(Main::method);

Method reference syntax for a static method

ContainingClass::staticMethodName

Method reference syntax for a instance method

containingObject::instanceMethodName

**Solution**

interface SalesItem {

    void sell();

}

class Main {

    public static void main(String [] args) {

        class EnergyDrink implements SalesItem {

            @Override

            public void sell() {

                System.out.println("EnergyDrink sold!");

            }

        }

        EnergyDrink ed = new EnergyDrink();

        sell(ed);

        class Dog implements SalesItem {

            @Override

            public void sell() {

                System.out.println("Doggie sold!");

            }

        }

        Dog spot = new Dog();

        sell(spot);

        SalesItem x = new SalesItem() {

            @Override

            public void sell() {

                System.out.println("Nameless item sold!");

            }

        };

        sell(x);

        sell( () -> System.out.println("lambda sold!"));

        sell(() -> method());

        sell(Main::method);

    }

    public static void method() {

        System.out.println("method reference sold");

    }

    public static void sell(SalesItem item) {

        item.sell();

    }

}