

# Menu Driven Apps

Loops, the switch statement, and menus

---

Produced by: Dr. Siobhán Drohan  
Mr. Colm Dunphy  
Mr. Diarmuid O'Connor  
Dr. Frank Walsh



Waterford Institute of Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics  
<http://www.wit.ie/>

# Topics list

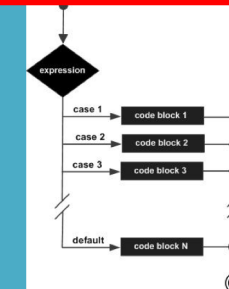
## 1. Loops

- while, for, for each
- Loop Control Variables (**LCV**)
- Arrays and **counter controlled** loops
- Arrays and **sentinel** based loops
- Arrays and **flag**-based loops

LOOPS  
MARKETING



## 2. switch statement



## 3. Menus

- A simple menu using switch.
- adding a menu to Shop v3.0.



# Recap - Loop Control Variable

1. Initialise

2. Test  
i.e. boolean condition

```
public static void simpleWhile() {
    int i = 0;
    while (i < 10)
    {
        System.out.println("Hello");
        i++;
    }
}
```

3. Update directly before end of loop

This loop is a **counter-controlled** while loop

# Topics list

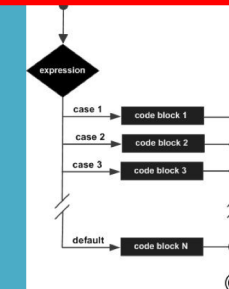
## 1. Loops

- while, for, for each
- Loop Control Variables (LCV)
- Arrays and **counter controlled** loops
- Arrays and **sentinel** based loops
- Arrays and **flag**-based loops

LOOPS  
MARKETING



## 2. switch statement



## 3. Menus

- A simple menu using switch.
- adding a menu to Shop v3.0.



# Recap - Counter-Controlled Loops

---



- Sometimes we know when we are coding i.e. **compile-time**, **how many** inputs we will have.

- See example 1



- Other times, we find out at **run-time** **how many** inputs we have

- See example 2



# Recap - Counter-Controlled **for** Loop: number of inputs known at **compile-time**



## Example 1

```
public static void loopWithArrayExample() {  
    int[] numbers = new int[10];    //array is a local variable  
    int sum = 0;  
  
    for (int i = 0; i < 5; i++)  
    {  
        System.out.print ("Please enter a number : ");  
        numbers[i] = input.nextInt();  
        sum += numbers[i];  
    }  
  
    System.out.println("The sum of the numbers you typed in is : " + sum);  
}
```



# Recap - Counter-Controlled **for** Loop: number of inputs known at **run-time**



Example 2

```
public static void loopWithArrayVarSizeExample() {  
    int[] numbers = null;  
    int numNumbers = 0;  
    int sum = 0;  
  
    System.out.print ("How many numbers would you like to enter?  
    numNumbers = input.nextInt();  
    numbers = new int[numNumbers];  
  
    for (int i = 0; i < numNumbers; i++)  
    {  
        System.out.print ("Please enter a number : ");  
        numbers[i] = input.nextInt();  
        sum += numbers[i];  
    }  
  
    System.out.println("The sum of the numbers you typed in is : " + sum);  
}
```



Here, we know at **run-time** how many inputs we have.

# Topics list

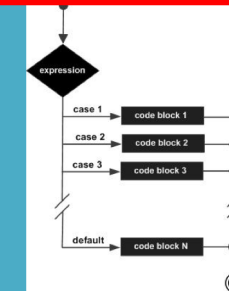
## 1. Loops

- while, for, for each
- Loop Control Variables (LCV)
- Arrays and **counter controlled** loops
- Arrays and **sentinel** based loops
- Arrays and **flag**-based loops

LOOPS  
MARKETING



## 2. switch statement



## 3. Menus

- A simple menu using switch.
- adding a menu to Shop v3.0.





# Sentinel-based loops

---



- Use this type of loop when you **DON'T know how many** inputs you will have.
- The *end of input* is signalled by a special value.
  - e.g.
    - if you are calculating the ‘average of ages of people in the room’:
      - write a program that will continually take in ages until, say, **-1** is entered.
    - **-1** would be the **sentinel value**.

# Structure

---



- Concept of ***Loop Control Variable*** is vital here.
- The loop continuation is solely based on the input, so the variable containing the information is the Loop Control Variable.
- Initialise the Loop Control Variable before entry into the loop.
- Remember to ‘update the Loop Control Variable’ just before the end of the loop.

# Try this exercise



- Write a loop to read in and add up a set of integers. Keep going until the value '-1' is inputted.
- What is your Loop Control Variable?

*Note: for this exercise, don't store the values in an array... we'll do that in a few slides time.*

# Solution



```
public static void sentinelWhileLoop()
{
    int sum = 0;

    System.out.print("Enter a number, -1 ends input: ");
    int n = input.nextInt();

    while (n != -1)
    {
        sum += n;
        System.out.print("Enter a number, -1 ends input: ");
        n = input.nextInt();
    }
    System.out.println("The total is: " + sum);
}
```

1. Initialise

2. LCV Condition

3. Update LCV directly  
before end of loop

# Next step in the exercise

---



- We need to record how many inputs have happened.
- Change the previous solution so that you know at the end, **how many** numbers have been inputted.
- At the end, print the sum and number of inputs.



# Code with number of inputs



```
public static void sentinelWhileLoopWithCounter()
{
    int sum = 0, counter = 0;

    System.out.print("Enter a number, -1 ends input: ");
    int n = input.nextInt();

    while (n != -1)
    {
        sum += n;
        System.out.print("Enter a number, -1 ends input: ");
        n = input.nextInt();
        counter++;
    }
    System.out.println("The total is: " + sum);
    System.out.println("The number of items entered is: " + counter);
}
```



# Try this now - using arrays

---



- Re-write the code on the previous slide, but **store the data in an array**.
  - NOTE:
    - Assume the max number of inputs possible is 100 (i.e. size of array).
- We also need to know
  - **how many inputs** actually happened.



# Solution – storing inputs



```
public static void sentinelWhileLoopWithArrays()
{
    int sum = 0, counter = 0, size = 100;
    int numbers [] = new int[size];

    System.out.print("Enter a number, -1 ends input: ");
    int n = input.nextInt();

    while (n != -1 && counter < size) //ensures that you don't go over max size of array
    {
        numbers[counter] = n;
        sum += n;
        System.out.print("Enter a number, -1 ends input: ");
        n = input.nextInt();
        counter++;
    }
    System.out.println("The total is: " + sum);
    System.out.println("The number of items entered is: " + counter);

    for (int i = 0; i < counter; i++)
    {
        System.out.println("    Number entered: " + numbers[i]);
    }
}
```





# Topics list

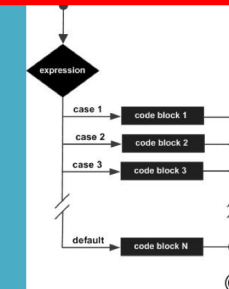
## 1. Loops

- while, for, for each
- Loop Control Variables (LCV)
- Arrays and **counter controlled** loops
- Arrays and **sentinel** based loops
- Arrays and **flag**-based loops

LOOPS  
MARKETING



## 2. switch statement



## 3. Menus

- A simple menu using switch.
- adding a menu to Shop v3.0.



# Flag-Based Loops

---



- These are used when you want to
  - examine a collection of data
  - to ***check for a property***.
  - Once a property has been established, it cannot be ‘unestablished’.
- *‘Once the flag is raised, it cannot be taken down’*

# Flag-Based Loops - example

---



- Given a populated array of numbers, cycle over the array to see if any numbers are odd.
- If you find:
  - At least one odd number,
    - print out to the user that there is at least one odd number.
  - No odd numbers,
    - inform the user of this.

# Solution: check if 'any numbers odd'



```
public static void flagBasedLoopWithArray()
```

```
{
```

```
    int numbers[] = {4,6,8,7,10,12};
```

```
    boolean oddNumberInArray = false;
```

```
    for (int number : numbers)
```

```
    {
```

```
        if (number % 2 == 1)
```

```
        {
```

```
            oddNumberInArray = true;
```

```
        }
```

```
    }
```

```
    if (oddNumberInArray == true)
```

```
    {
```

```
        System.out.println("There is at least one odd number in the array.");
```

```
    }
```

```
    else
```

```
    {
```

```
        System.out.println("There is NO odd number in the array.");
```

```
    }
```

```
}
```

// For each number in the array numbers

//if number mod 2 (remainder after division by 2)

# Better code...



```
public static void flagBasedLoopWithArray()
{
    int numbers[] = {4,6,8,7,10,12};
    boolean oddNumberInArray = false;

    for (int number : numbers)
    {
        if (number % 2 == 1)
        {
            oddNumberInArray = true;
        }
    }

    if (oddNumberInArray)
    {
        System.out.println("There is at least one odd number in the array.");
    }
    else
    {
        System.out.println("There is NO odd number in the array.");
    }
}
```

Use of boolean variable in condition

if (oddNumberInArray)

*What about having a  
flag-based loop  
in a method  
with a boolean return type?*

## Method Definition - boolean return type

```
public static boolean flagBasedLoopWithArrayReturn()
{
    int numbers[] = {4,6,8,7,10,12};
    boolean oddNumberInArray = false;

    for (int number : numbers)
    {
        if (number % 2 == 1)
        {
            oddNumberInArray = true;
        }
    }

    return oddNumberInArray;
}
```

// For each number in the array numbers

//if number mod 2 (remainder after division by 2)

// return true or false

## Calling the method - handling the returned boolean

```
if (flagBasedLoopWithArrayReturn())
    System.out.println("There is at least one odd number in the array");
else
    System.out.println("There is NO odd number in the array");
```

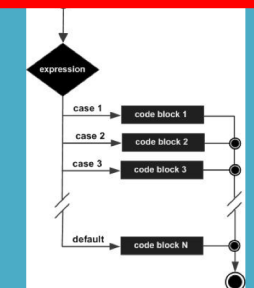
# Topics list

## 1. Loops

- while, for, for each
- Loop Control Variables (**LCV**)
- Arrays and **counter controlled** loops
- Arrays and **sentinel** based loops
- Arrays and **flag**-based loops



## 2. switch statement



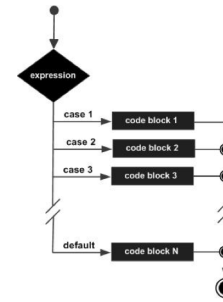
## 3. Menus

- A simple menu using switch.
- adding a menu to Shop v3.0.



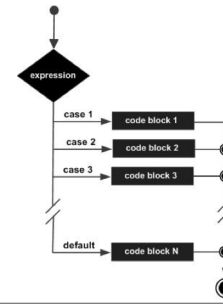


# The **switch** statement



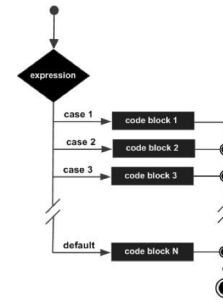
- The switch statement works in exactly the same way as a **set of if** statements, but is more compact and readable.
- The *switch statement* switches on a single **value** to one of an arbitrary number of **cases**.
- Two possible patterns of use are...

# The `switch` statement – pattern one



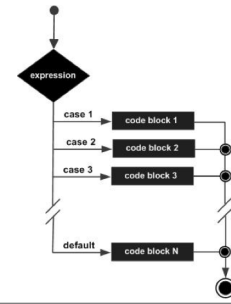
```
switch(expression) {  
    case value: statements;  
                break;  
    case value: statements;  
                break;  
    further cases possible  
    default: statements;  
            break;  
}
```

# The switch statement – pattern two



```
switch(expression) {  
    case value1:  
    case value2:  
    case value3:  
        statements;  
        break;  
    case value4:  
    case value5:  
        statements;  
        break;  
    further cases possible  
    default:  
        statements;  
        break;  
}
```

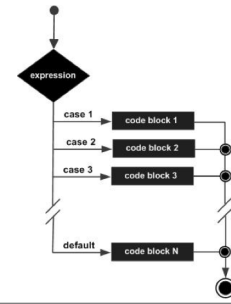
# The switch statement



- A *switch* statement can have any number of **case** labels.

```
switch(expression) {  
  case value: statements;  
              break;  
  case value: statements;  
              break;  
  further cases possible  
  default: statements;  
           break;  
}
```

# The switch statement



The **break** statement after every case is needed, otherwise the execution “falls through” into the next label’s statements.

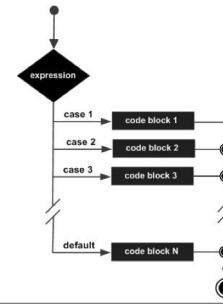
Pattern **two** makes use of this.

All three of the first values (cases) will execute the first *statements* section,

Values (cases) four and five will execute the second *statements* section.

```
switch(expression) {  
  case value1:  
  case value2:  
  case value3:  
    statements;  
    break;  
  case value4:  
  case value5:  
    statements;  
    break;  
  further cases possible  
  default:  
    statements;  
    break;  
}
```

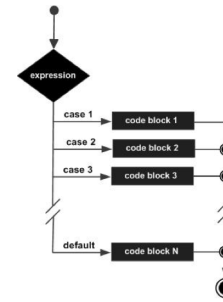
# The switch statement



- The **default** case is optional.
- If no default is given, it may happen that no case is executed.

```
switch(expression) {  
    case value: statements;  
                break;  
    case value: statements;  
                break;  
    further cases possible  
    default: statements;  
            break;  
}
```

# The switch statement

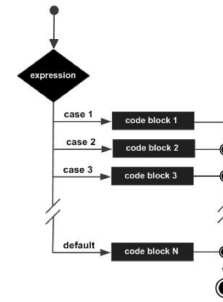


- The **break** statement after the default  
(or the last case, if there is no default)

is not needed but is considered good style.

```
switch(expression) {  
    case value: statements;  
                break;  
    case value: statements;  
                break;  
    further cases possible  
    default: statements;  
            break;  
}
```

# The switch statement



- Pre Java 7,  
the **expression** used to switch on,

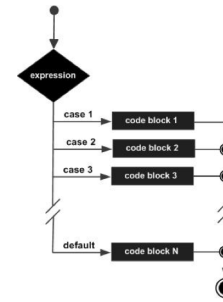
and the case **labels (value)** are  
**char** or **int**.

- Post Java 7,  
you can also switch on **String**.

```
switch(expression) {  
    case value: statements;  
                break;  
    case value: statements;  
                break;  
    further cases possible  
    default: statements;  
            break;  
}
```

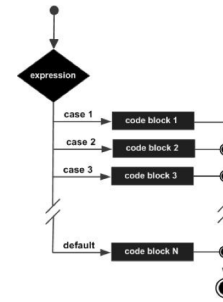


# The switch statement – **int** example



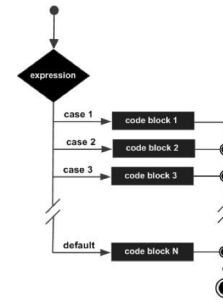
```
switch(day) {  
    case 1: dayString = "Monday";  
            break;  
    case 2: dayString = "Tuesday";  
            break;  
    case 3: dayString = "Wednesday";  
            break;  
    case 4: dayString = "Thursday";  
            break;  
    case 5: dayString = "Friday";  
            break;  
    case 6: dayString = "Saturday";  
            break;  
    case 7: dayString = "Sunday";  
            break;  
    default: dayString = "invalid day";  
            break;  
}
```

# The switch statement – **char** example



```
switch (group){
    case 'A':
        System.out.println("10.00 a.m ");
        break;
    case 'B':
        System.out.println("1.00 p.m ");
        break;
    case 'C':
        System.out.println("11.00 a.m ");
        break;
    default:
        System.out.println("Enter option A, B or C only!");
}
```

# The switch statement – **String** example



```
switch(dow.toLowerCase()) {  
    case "mon":  
    case "tue":  
    case "wed":  
    case "thu":  
    case "fri":  
        goToWork();  
        break;  
    case "sat":  
    case "sun":  
        stayInBed();  
        break;  
}
```

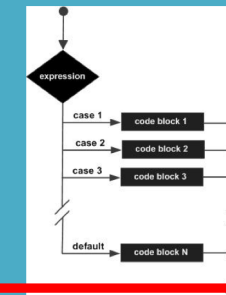
# Topics list

## 1. Loops

- while, for, for each
- Loop Control Variables (**LCV**)
- Arrays and **counter controlled** loops
- Arrays and **sentinel** based loops
- Arrays and **flag**-based loops



## 2. switch statement

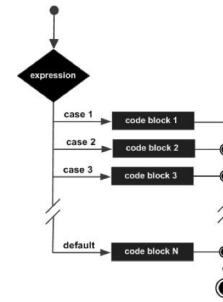


## 3. Menus

- A simple menu using switch.
- adding a menu to Shop v3.0.



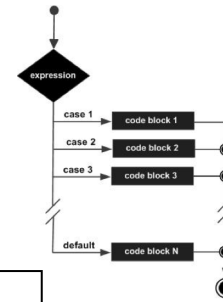
# A simple menu using switch



```
public void run()
{
    System.out.println("Choose a number between 1 and 3");
    int choice = input.nextInt();

    switch(choice)
    {
        case 1:
            System.out.println("You chose 1");
            break;
        case 2:
            System.out.println("You chose 2");
            break;
        case 3:
            System.out.println("You chose 3");
            break;
        default:
            System.out.println("You chose an invalid number");
            break;
    }
}
```

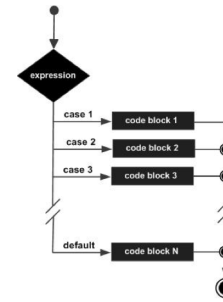
# Now loop on the switch statement



```
public void run()
{
    System.out.println("Choose a number between 1 and 3");
    int choice = input.nextInt();
    while (choice != 0)
    {
        switch(choice)
        {
            case 1:
                System.out.println("You chose 1");
                break;
            case 2:
                System.out.println("You chose 2");
                break;
            case 3:
                System.out.println("You chose 3");
                break;
            default:
                System.out.println("You chose an invalid number");
                break;
        }
        System.out.println("Choose a number between 1 and 3");
        choice = input.nextInt();
    }
}
```

Note the use of the Loop Control Variable

# This gives the following output



```
Choose a number between 1 and 3
2
You chose 2
Choose a number between 1 and 3
3
You chose 3
Choose a number between 1 and 3
9
You chose an invalid number
Choose a number between 1 and 3
0
```

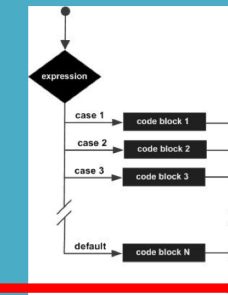
# Topics list

## 1. Loops

- while, for, for each
- Loop Control Variables (**LCV**)
- Arrays and **counter controlled** loops
- Arrays and **sentinel** based loops
- Arrays and **flag**-based loops



## 2. switch statement



## 3. Menus

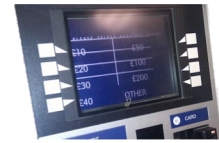
- A simple menu using switch.
- adding a menu to Shop v3.0.





# Adding a basic menu to Shop...

---



```
Shop Menu
-----
  1) Add a Product
  2) List the Products
  0) Exit
==>>
```

# Menu to be displayed...

## 1. mainMenu()



Driver

```
/**
 * mainMenu() - This method displays the menu for the application,
 * reads the menu option that the user entered and returns it.
 *
 * @return      the users menu choice
 */
private int mainMenu() {
    System.out.println("Shop Menu");
    System.out.println("-----");
    System.out.println(" 1) Add a Product");
    System.out.println(" 2) List the Products");
    System.out.println(" 0) Exit");
    System.out.print("==>> ");
    int option = input.nextInt();
    return option;
}
```

# Handling the menu input...

## 2. runMenu()



Driver

```
private void runMenu() {  
    int option = mainMenu();  
  
    while (option != 0) {  
        switch (option) {  
            case 1:    addProduct();  
                break;  
            case 2:    System.out.println(store.listProducts());  
                break;  
            default:   System.out.println("Invalid option entered: " + option);  
                break;  
        }  
  
        //pause the program so the user can read what we just printed to the terminal window  
        System.out.println("\nPress any key to continue...");  
        input.nextLine();  
        input.nextLine();    //this second read is required - bug in Scanner class;  
                             //a String read is ignored straight after reading an int.  
  
        //display the main menu again  
        option = mainMenu();  
    }  
}
```

# Calling the menu on startup... 3. Driver()



## Driver

```
public class Driver{  
  
    private Scanner input = new Scanner(System.in);  
    private Store store;  
  
    public static void main(String[] args) {  
        Driver c = new Driver();  
    }  
  
    public Driver()  
    {  
        store = new Store();  
        runMenu();  
    }  
}
```

### Shop Menu

```
-----  
1) Add a Product  
2) List the Products  
0) Exit  
==>>
```

# A more evolved Shop menu...

---



## Shop Menu

-----

- 1) Add a Product
- 2) List the Products

-----

- 3) List the cheapest product
- 4) List the products in our current product line
- 5) Display average product unit cost
- 6) List products that are more expensive than a given price

0) Exit

==>>

# 1. mainMenu()

Driver

```
/**
 * mainMenu() - This method displays the menu for the application,
 * reads the menu option that the user entered and returns it.
 *
 * @return      the users menu choice
 */
private int mainMenu(){
    System.out.println("Shop Menu");
    System.out.println("-----");
    System.out.println("  1) Add a Product");
    System.out.println("  2) List the Products");
    System.out.println("-----");
    System.out.println("  3) List the cheapest product");
    System.out.println("  4) List the products in our current product line");
    System.out.println("  5) Display average product unit cost");
    System.out.println("  6) List products that are more expensive than a given price");
    System.out.println("  0) Exit");
    System.out.print("==>> ");
    int option = input.nextInt();
    return option;
}
```

## 2. runMenu()

Driver

```
private void runMenu(){
    int option = mainMenu();
    while (option != 0){

        switch (option){
            case 1:    addProduct();
                      break;
            case 2:    System.out.println(store.listProducts());
                      break;
            case 3:    System.out.println(store.cheapestProduct());
                      break;
            case 4:    System.out.println(store.listCurrentProducts());
                      break;
            case 5:    System.out.println(store.averageProductPrice());
                      break;
            case 6:    System.out.print("Enter the price barrier: ");
                      double price = input.nextDouble();
                      System.out.println(store.listProductsAboveAPrice(price));
                      break;
            default:   System.out.println("Invalid option entered: " + option);
                      break;
        }

        //pause the program so the user can read what we just printed to the terminal window
        System.out.println("\nPress any key to continue...");
        input.nextLine();
        input.nextLine();    //this second read is required - bug in Scanner class;
                             //a String read is ignored straight after reading an int.

        //display the main menu again
        option = mainMenu();
    }
}
```

### 3. Driver()

## Driver

```
public class Driver{

    private Scanner input = new Scanner(System.in);
    private Store store;

    public static void main(String[] args) {
        Driver c = new Driver();
    }

    public Driver()
    {
        store = new Store();
        runMenu();
    }
}
```

#### Shop Menu

-----

- 1) Add a Product
- 2) List the Products

-----





- 3) List the cheapest product
- 4) List the products in our current product line
- 5) Display average product unit cost
- 6) List products that are more expensive than a given price
- 0) Exit

==>>



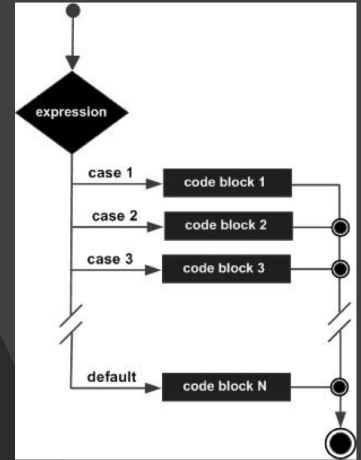
# Summary

## 1. Loops

- Recap of
  - while, for, for each
- Recap
  - Loop Control Variables (LCV)
- Arrays and **counter controlled** loops
  - Known at compile time 
  - Known at run time 
- Arrays and **sentinel** based loops
  - Special value e.g. -1 
- Arrays and **flag**-based loops
  - Test for condition, raise flag if true, can't take it down 

## 2. switch statement

```
switch(day) {  
  case 1: dayString = "Monday";  
          break;  
  case 2: dayString = "Tuesday";  
          break;  
  case 3: dayString = "Wednesday";  
          break;  
  case 4: dayString = "Thursday";  
          break;  
  case 5: dayString = "Friday";  
          break;  
  case 6: dayString = "Saturday";  
          break;  
  case 7: dayString = "Sunday";  
          break;  
  default: dayString = "invalid day";  
           break;  
}
```



## 3. Menus

- A simple menu using switch
- added a menu to **Shop v3.0**.
  - 3 changes to Driver class
    - mainMenu()
    - runMenu()
    - Driver()



# Next

---

- Unit Testing
  - Pre-requisite for next assignment
  - JUnit
  - TDD
    - Test driven development
- CRUD
- Debugging

# Questions?

---

