

Java Definitions

- **Object** - Objects have states and behaviours. Example: A dog has states - colour, name, breed as well as behaviours -wagging, barking, eating. An object is an instance of a class.
- **Class** - A class can be defined as a template/ blue print that describes the behaviours/states that object of its type support.
- **Methods** - A method is basically a behaviour. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** - Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Data Types

- Primitive Data Types
- Reference/Object Data Types



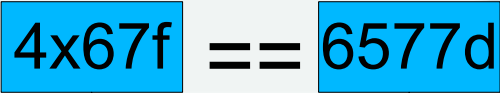
```
byte b = 68;  
char c = 'A';  
int i = 5;  
float f = 4.5f;  
double d = 7.889;  
long l = 5000l;  
boolean b = true;
```

```
Integer int = new Integer(5);  
Double dou = new Double (6.7);  
Boolean bool = new Boolean.TRUE;
```

Primitive

- Primitive data types are a direct value in memory
- `int a = 5;` 5
- `int b = 5;` 5
- `a == b ?` 5 == 5
- So it's true, a does equal b

Object

- An object is a reference to some data
- Integer x = new Integer(5); 
- Integer y = new Integer(5); 
- if(x == y) → FALSE 
- if(x.intValue() == y.intValue()) → TRUE

Class

- An Object that be instantiated (generally)

```
public class Dog{  
    String breed;  
    int ageC  
    String color;  
  
    void barking(){  
    }  
  
    void hungry(){  
    }  
  
    void sleeping(){  
    }  
}
```

Classes Have

- **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables:** Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables:** Class variables are variables declared with in a class, outside any method, with the static keyword.

Classes also have COMMENTS!

```
// This is a one line comment
```

```
/*
```

```
 * This is a block comment
```

```
*/
```

- **The comment says what you WANT/THINK the code to do/does**

- **DO NOT DO THIS:**

```
// add 1 to i
```

```
i = i + 1;
```

- I know it adds 1 to i, but I have no idea **why**, or if it is correct thing to do

Classes also have COMMENTS!

- DO THIS:

```
// I want to make the robot turn left so I'm  
increasing the angle
```

```
i = i + 1;
```


- Now I know why you're doing it
- I can also spots that should we be **increasing** the angle to go right?

Variables Scope

```
public class Dog{  
    static String NOISE = "Class variable";  
    String breed = "instance variable";  
  
    void barking(){  
        String var = "local var";  
    }  
}
```

Variables Scope

```
public class Dog{  
    static String NOISE = "Class variable";  
    String breed = "instance variable";  
  
    void barking(){  
        String breed = "local var";  
        System.out.println(breed);  
        System.out.println(this.breed);  
    }  
}
```



Telling Java to use the instance, rather
Than the local variable

Access Modifiers

- **public** – everyone can access it
- **private** – only this class can access it
- **protected** – this class and classes that extend it can access it
- **<nothing>** - default, other classes within the same package can access it
- Set variables to the lowest visibility that you require, ideally everything is private

Non-Access Modifiers

- **final** – The variable cannot be changed at run-time
- **static** – The variable only exists once, all Objects will refer to the same variable (be careful with this)
- **abstract** – The method must be implemented by a subclass, it does not yet exist
- **synchronized** – The method can only be run once at a time. Some threads may have to wait

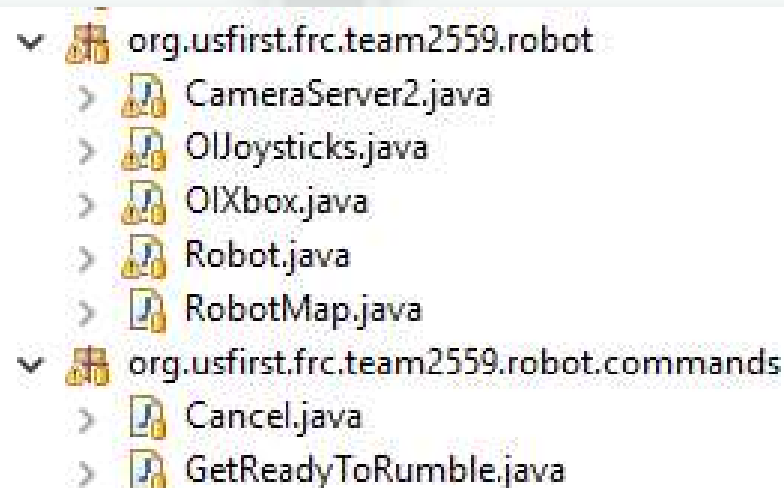
Constructors

- Objects can be constructed multiple ways depending on the data available
- `new Puppy();`
- `new Puppy("Sammy");`

```
public class Puppy{  
    public Puppy(){  
    }  
  
    public Puppy(String name){  
        // This constructor has one parameter, name.  
    }  
}
```

Imports

- To use a class from another class you need to tell Java where to find it
- Robot.java:
- `import org.usfirst.frc.team2559.robot.commands.Cancel;`



Arithmetic Operators

Operator and Example

1 + (Addition)

Adds values on either side of the operator

Example: $A + B$ will give 30

2 - (Subtraction)

Subtracts right hand operand from left hand operand

Example: $A - B$ will give -10

3 * (Multiplication)

Multiplies values on either side of the operator

Example: $A * B$ will give 200

4 / (Division)

Divides left hand operand by right hand operand

Example: B / A will give 2

$A = 10$

$B = 20$

Arithmetic Operators

5 % (Modulus)

Divides left hand operand by right hand operand and returns remainder

Example: B % A will give 0

6 ++ (Increment)

Increases the value of operand by 1

Example: B++ gives 21

7 -- (Decrement)

Decreases the value of operand by 1

Example: B-- gives 19

A = 10

B = 20

Relational Operators

SR.NO Operator and Description

1 == (equal to)

Checks if the values of two operands are equal or not, if yes then condition becomes true.

Example: (A == B) is not true.

2 != (not equal to)

Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

Example: (A != B) is true.

A = 10

B = 20

3 > (greater than)

Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.

Example: (A > B) is not true.

4 < (less than)

Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.

Example: (A < B) is true.

Relational Operators

5 \geq (greater than or equal to)

Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

Example (A \geq B) is not true.

6 \leq (less than or equal to)

Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

example(A \leq B) is true.

A = 10

B = 20

Logical Operators

Operator	Description
----------	-------------

1 && (logical and)	
-------------------------	--

Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.

Example (A && B) is false.

2 (logical or)	
------------------------	--

Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.

Example (A || B) is true.

3 ! (logical not)	
------------------------	--

Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

Example !(A && B) is true.

A = true

B = false

Assignment Operators

Operator and Description

1 =

Simple assignment operator, Assigns values from right side operands to left side operand.

Example: $C = A + B$ will assign value of $A + B$ into C

2 +=

Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.

Example: $C += A$ is equivalent to $C = C + A$

3 -=

Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.

Example: $C -= A$ is equivalent to $C = C - A$

4 *=

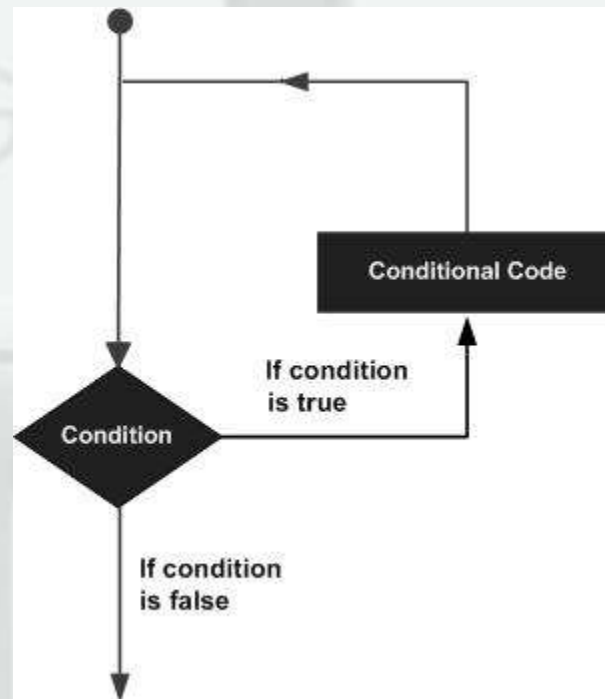
Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand.

Example: $C *= A$ is equivalent to $C = C * A$

MAKE YOU CODE CLEAR!

Looping

- Keep doing something until a condition is met



Loop Types

while loop

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

for loop

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

do...while loop

Like a while statement, except that it tests the condition at the end of the loop body

- Try to avoid do..while
- You have to scroll down the code to see what the loop is

Loop Control Statements

break statement

Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.

continue statement

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

- Ideally don't use above
- boolean shouldExit = false
- while (!shouldExit) {

Enhanced Loop

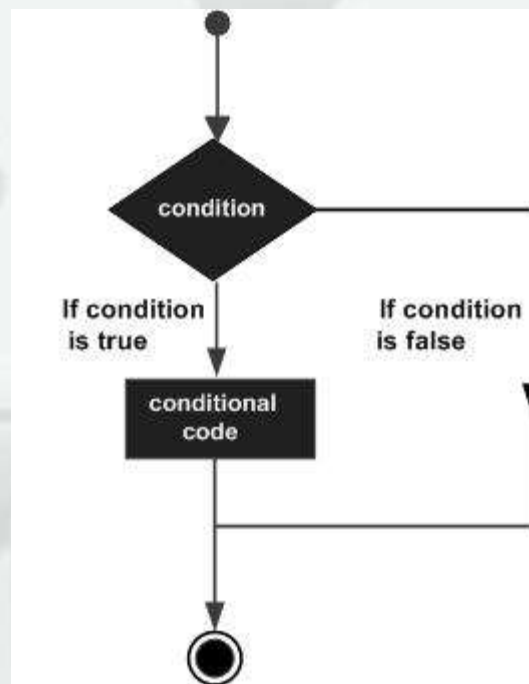
- The lazy way to loop through everything
- Quick to code, but less control

```
int [] numbers = {10, 20, 30, 40, 50};
```

```
for(int x : numbers ){  
    System.out.print( x );  
    System.out.print(",");  
}
```

Decision Making

- If (some condition) then...



Decision Making

if statement

An if statement consists of a boolean expression followed by one or more statements.

if...else statement

An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

nested if statements

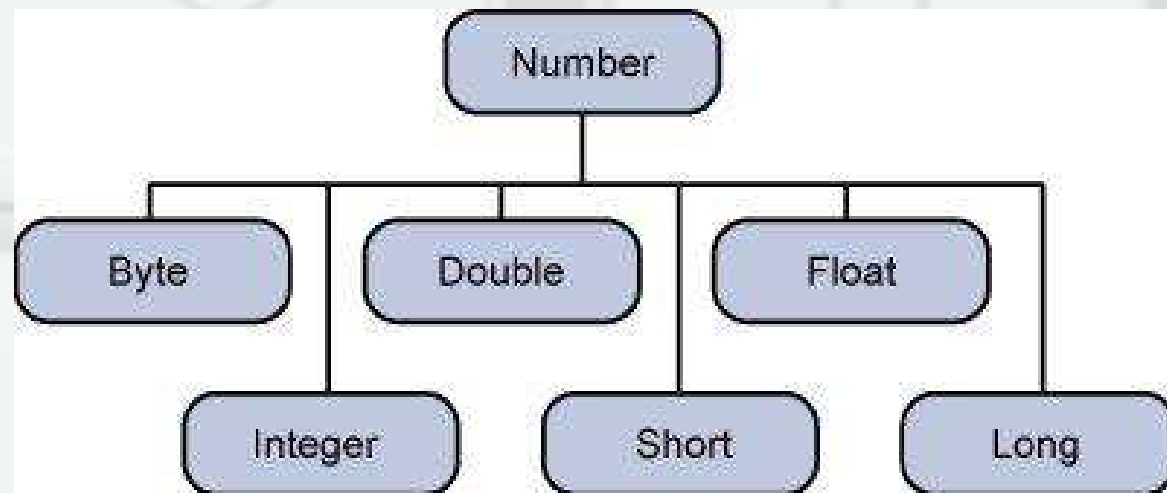
You can use one if or else if statement inside another if or else if statement(s).

switch statement

A switch statement allows a variable to be tested for equality against a list of values.

Numbers

- There are lots of types of Numbers
 - ◆ Integer, Long, and Double are most common



Java Boxing

- Java will convert primitives to Objects for you – be careful with this!

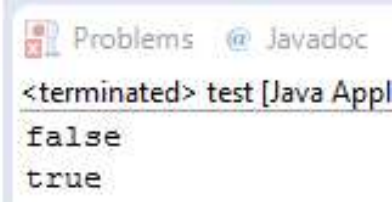
```
public class Test {  
  
    public static void main(String args[]) {  
        Integer x = 5; // boxes int to an Integer object  
        x = x + 10; // unboxes the Integer to a int  
        System.out.println(x);  
    }  
}
```

Java Boxing

- In the first case we create two Integers, each with 10 in them
- In the second case there is one global constant Integer(10) that both are pointing to

```
Integer x = new Integer(10);  
Integer y = new Integer(10);  
System.out.println(x == y);
```

```
Integer a = 10;  
Integer b = 10;  
System.out.println(a == b);
```



```
Problems @ Javadoc  
<terminated> test [Java Appl  
false  
true
```

Strings

- Internally it's an array of characters

```
public class StringDemo {  
  
    public static void main(String args[]) {  
        String palindrome = "Dot saw I was Tod";  
        int len = palindrome.length();  
        System.out.println( "String Length is : " + len );  
    }  
}
```

Strings

- They can be added (create a new String)
 - ◆ “Hello “ + “World!” = “Hello World”
- Or formatted (C style)
 - ◆ `String worldVar = “World!”;`
 - ◆ `String.format(“Hello %s”, worldVar);`

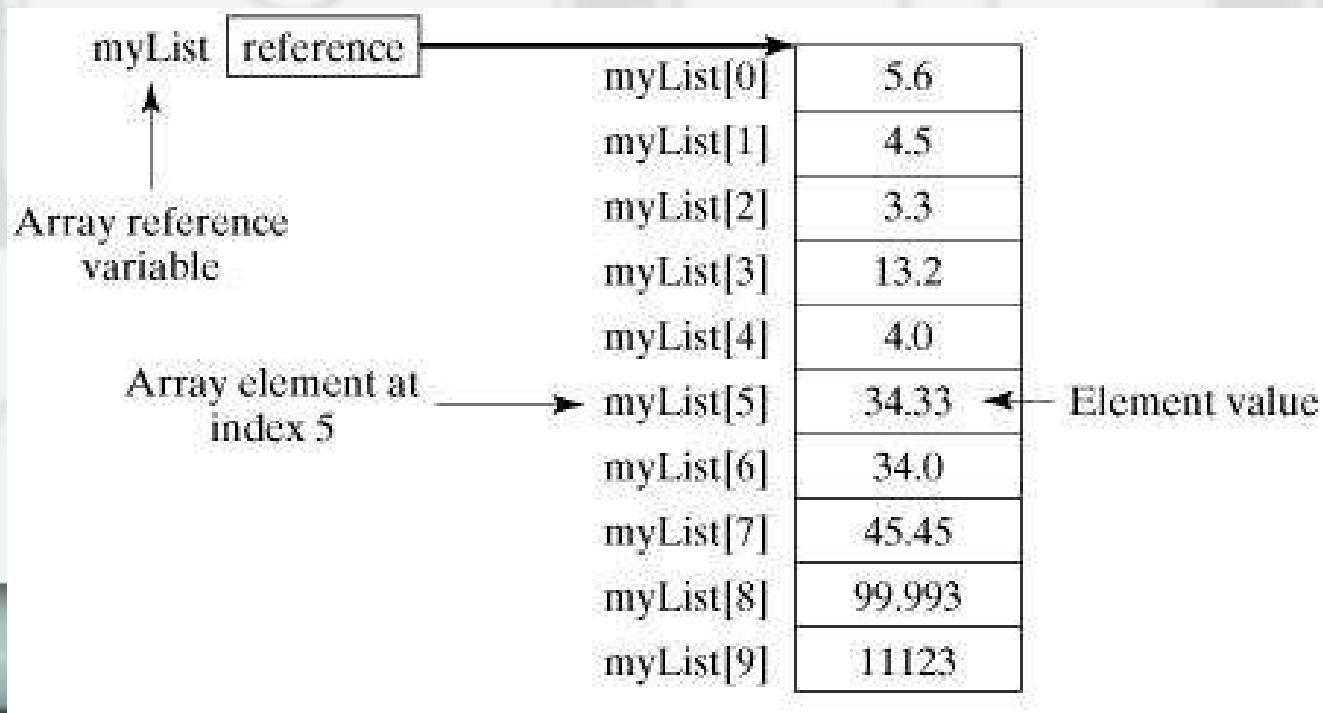
Arrays

- A group of items

`double[] myList;` `// preferred way.`

or

`double myList[];` `// works but not preferred way.`



Arrays

```
public static void main(String[] args) {  
    double[] myList = {1.9, 2.9, 3.4, 3.5};  
  
    // Print all the array elements  
    for (int i = 0; i < myList.length; i++) {  
        System.out.println(myList[i] + " ");  
    }  
    // Summing all elements  
    double total = 0;  
    for (int i = 0; i < myList.length; i++) {  
        total += myList[i];  
    }  
    System.out.println("Total is " + total);  
    // Finding the largest element  
    double max = myList[0];  
    for (int i = 1; i < myList.length; i++) {  
        if (myList[i] > max) max = myList[i];  
    }  
    System.out.println("Max is " + max);  
}
```

Array vs. ArrayList

- An ArrayList is a Java data type that has lots of methods already implemented

```
double[] myList = {1.9, 2.9, 3.4, 3.5};  
  
// Add an element  
double[] temp = new double[myList.length+1]  
  
for (int i = 0; i < myList.length; i++) {  
    temp[i] = myList[i];  
}  
temp[myList.length] = "99.9";  
  
MyList = temp;
```

```
ArrayList<Double> myList =  
    new ArrayList<Double>();  
  
myList.add(3.5);  
myList.add(99.9);
```