Software Programming Objectives

Part I

- Understand the “Atomic” element of the computer
- Understand the structure of a computer
- Understand types of computer languages
- Understand program control flow structures

Part II

- Understand classes and objects in JAVA
- Understand state machines in JAVA

Note: 1)These slides do not cover the entire JAVA language – only what is needed
2) Look for blue boxes for interactive instructions

Part I Computer Programming Objectives

- Understand the “Atomic” element of the computer
- Understand the structure of a computer
- Understand types of computer languages
- Understand program control flow structures

Some Definitions

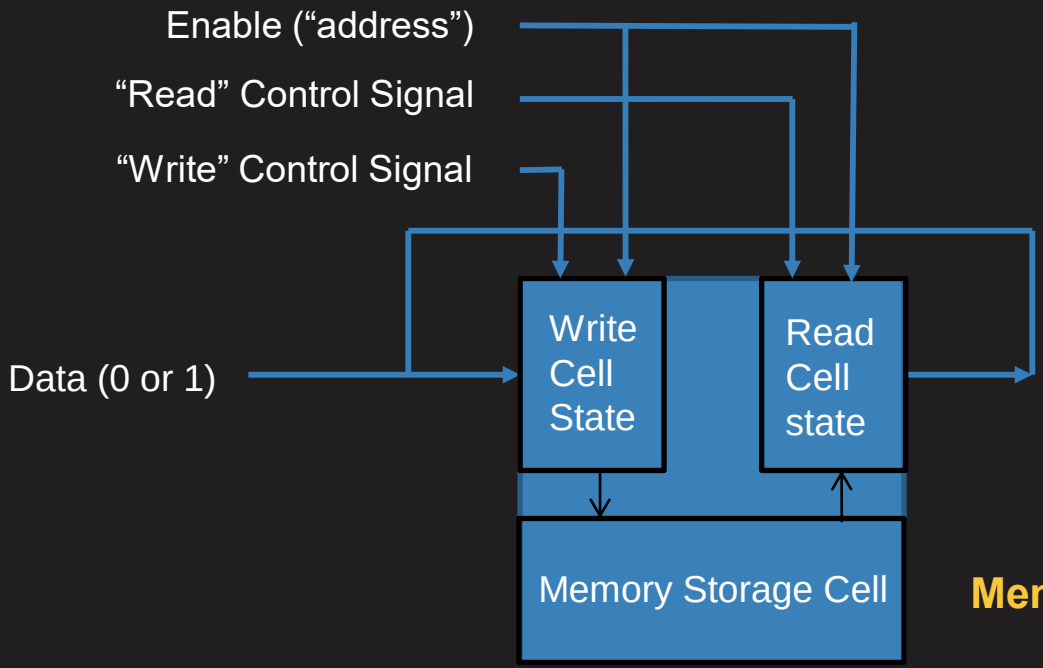
Definitions:
Computer: A computer is a machine that will execute a specific set of instructions in a well defined manner.

Computer Program: A pre-recorded list of instructions to perform a specific task when executed by the computer.

Programming Language: A programming language is a formal constructed human understandable language. The constructs of the language are then translated into instructions that a computer at the machine level can understand.

Programmer: The programmer develops instructions for the computer to do specific tasks.

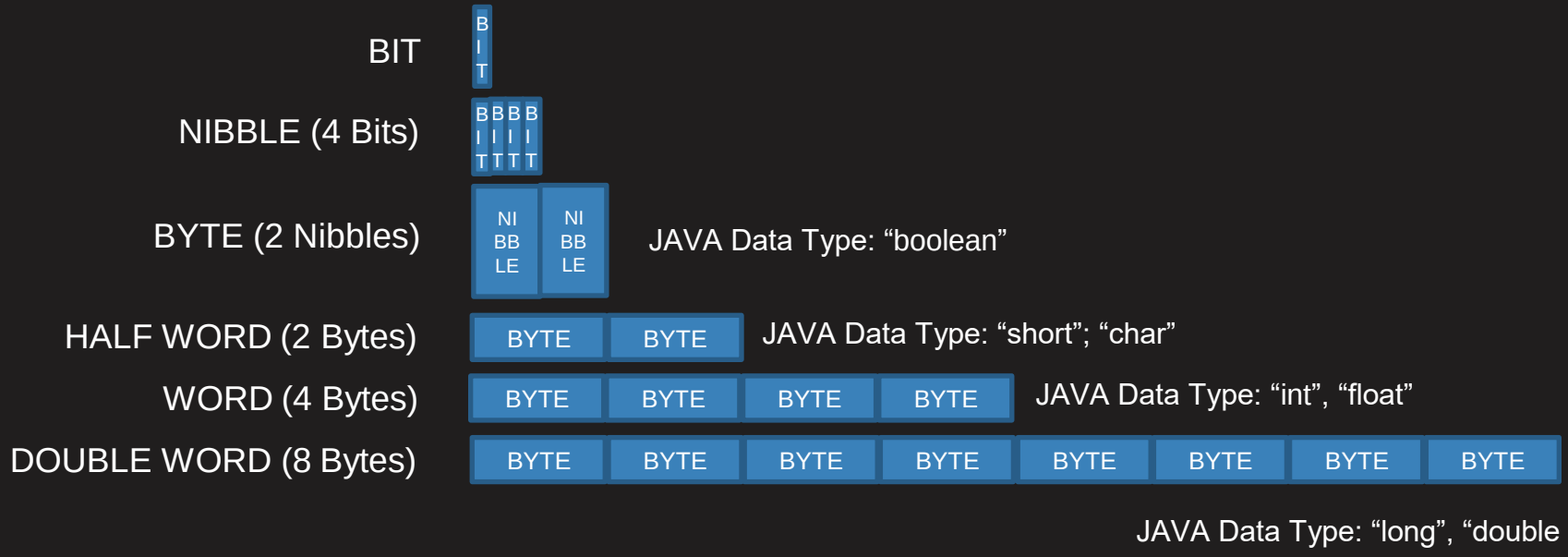
The “Atom” of a Computer: Memory Cell



Memory Cell => “BIT” (Binary digIT)

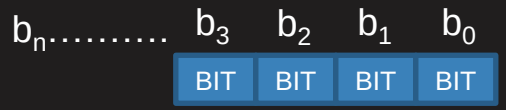
“Data” has two states: “0” or “1”

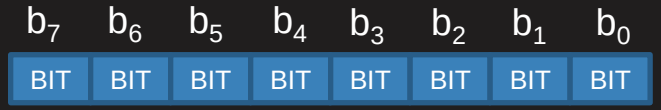
Memory Cell Configurations



BYTE (Smallest Addressable unit in a computer)

BIT Labeling: Numbers and Letters





DEC	HEX				
0	0x0	0	0	0	0
1	0x1	0	0	0	1
2	0x2	0	0	1	0
3	0x3	0	0	1	1
4	0x4	0	1	0	0
5	0x5	0	1	0	1
6	0x6	0	1	1	0
7	0x7	0	1	1	1
8	0x8	1	0	0	0
9	0x9	1	0	0	1
10	0xA	1	0	1	0
11	0xB	1	0	1	1
12	0xC	1	1	0	0
13	0xD	1	1	0	1
14	0xE	1	1	1	0
15	0xF	1	1	1	1

0x – Notation that number is Hexadecimal

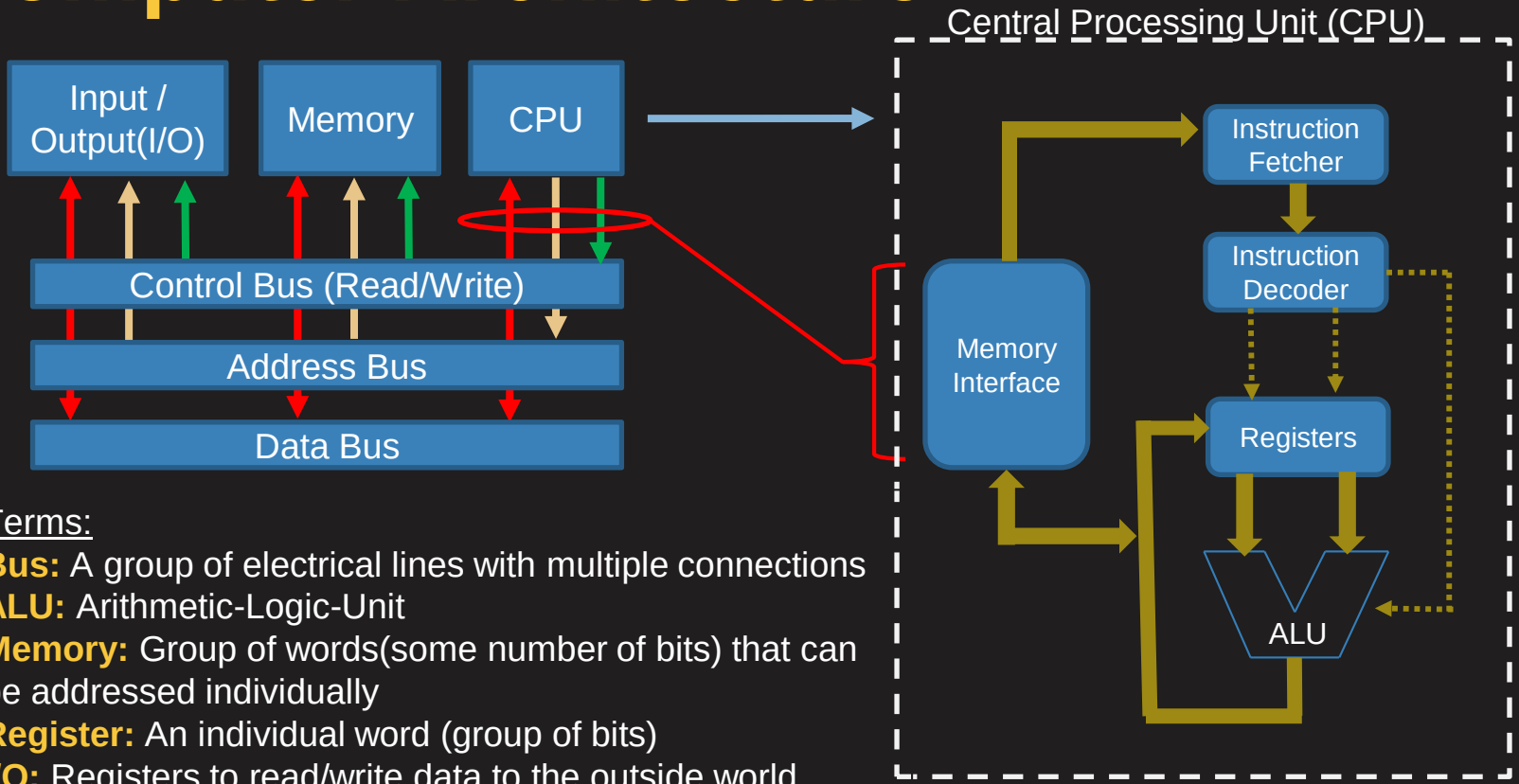
ASCII (8bit) codes to represent numbers and letters
For example:

	ASCII Code Dec	Hex
“A”	65	0x41
“a”	97	0x61
“1”	49	0x31
“2”	50	0x32

Etc…

“String”	Code Dec	Hex
“D”	68	0x44
“o”	111	0x6F
“g”	103	0x67

Computer Architecture



Terms:

Bus: A group of electrical lines with multiple connections
ALU: Arithmetic-Logic-Unit
Memory: Group of words(some number of bits) that can be addressed individually
Register: An individual word (group of bits)
I/O: Registers to read/write data to the outside world

Different Ways You Can Program a Computer

Machine Code:

- Toggle in 1's and 0's that a computer understands.
- (Done on the first computers in the 1950's-1960's)

Assembly Language(ASM)

- Enter in a text editor a mnemonic for each computer instruction.
- This is then put through a compiler to generate the 1's and 0's a computer understands.
- (This was done in the 1960's-1970's)

High Level Language(HLL)

- Enter in a text editor in a more natural language complex instruction forms(e.g.if-then-else, for loop,etc)
- This is then put through a compiler to generate the 1's and 0's a computer understands.
- (This was done in the 1980's-1990's)

Today

- We use an Integrated Development Environment(IDE) to develop our program instructions, compile our code and run our code.

Computer Instruction Language Levels

Machine Code:

- Machine code is a series of instructions that are hard coded in the CPU to do specific tasks.
- In a series of steps the CPU instruction engine will point to a memory location, fetches an instruction, decodes it, and executes it. Then the process is started again pointing to the next instruction in memory.

Assembly Language(ASM)

- Assembly language has mnemonics that represent every hard coded instruction in the computer
- Each mnemonic has an operation code(op code) and an operand.
- An assembler translates the mnemonic codes to machine codes for the computer to execute.
- Each manufacture's computer has its own set of mnemonics explicit to that computer.

For example:

ASM Lang. Mnemonic	Comment	**Assembler ➔**	Memory	Machine Code
.org $8020	Start at location 0x8020			
LDA, #$80;	Load Register A with 0x80		0x8020	0xA9 0x80
STA, $0315;	Store Register A at 0x0315		0x8022	0x8D 0x15 0x13
LDA, #$2D;	Load Register A with 0x2D		0x8025	0xA9 0x2D
STA, $0314;	Store Register A at 0x314		0x8027	0x8d 0x14 0x03

Computer Instruction Language Levels

High Level Language (HLL)

- C, C++, C#, JAVA, etc each have specific words and symbols(“syntax”) to allow natural language with an abstraction that is independent of the computer architecture
- A compiler will translate the program into machine code for a computer to run.
- An Integrated Development Environment (IDE) program helps the programmer to develop program code and then compile the code.

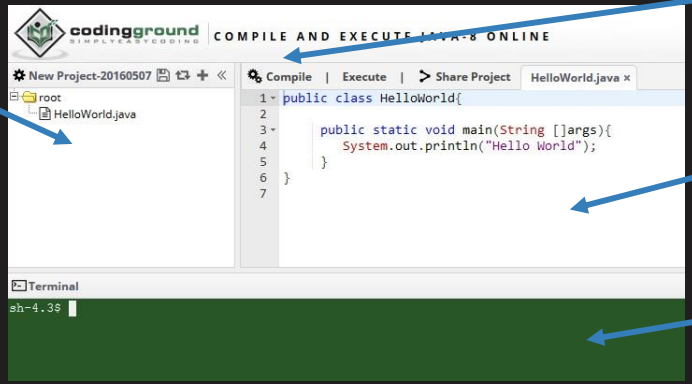
A web based tutorial IDE

Project program files

Compile and execute

Type your program text here

Console:
Where the output displays



http://www.tutorialspoint.com/compile_java_online.php

Program Start (“Hello World”) in JAVA

Open web site - http://www.tutorialspoint.com/compile_java_online.php

JAVA syntax for the overall program “HelloWorld”

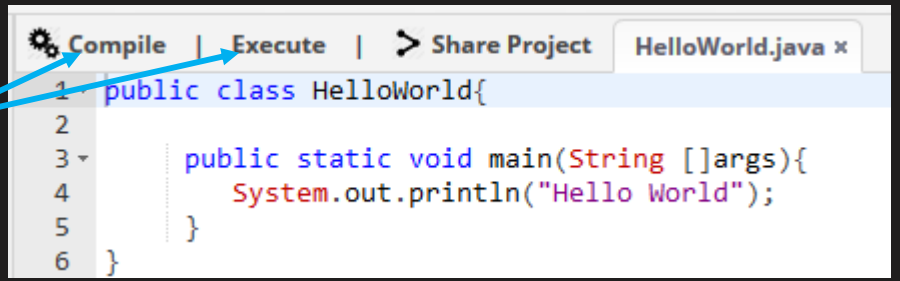
JAVA syntax for the program start

```
1  public class HelloWorld{
2
3      public static void main(String []args){
4          System.out.println("Hello World");
5      }
6  }
```

Notice the syntax braces

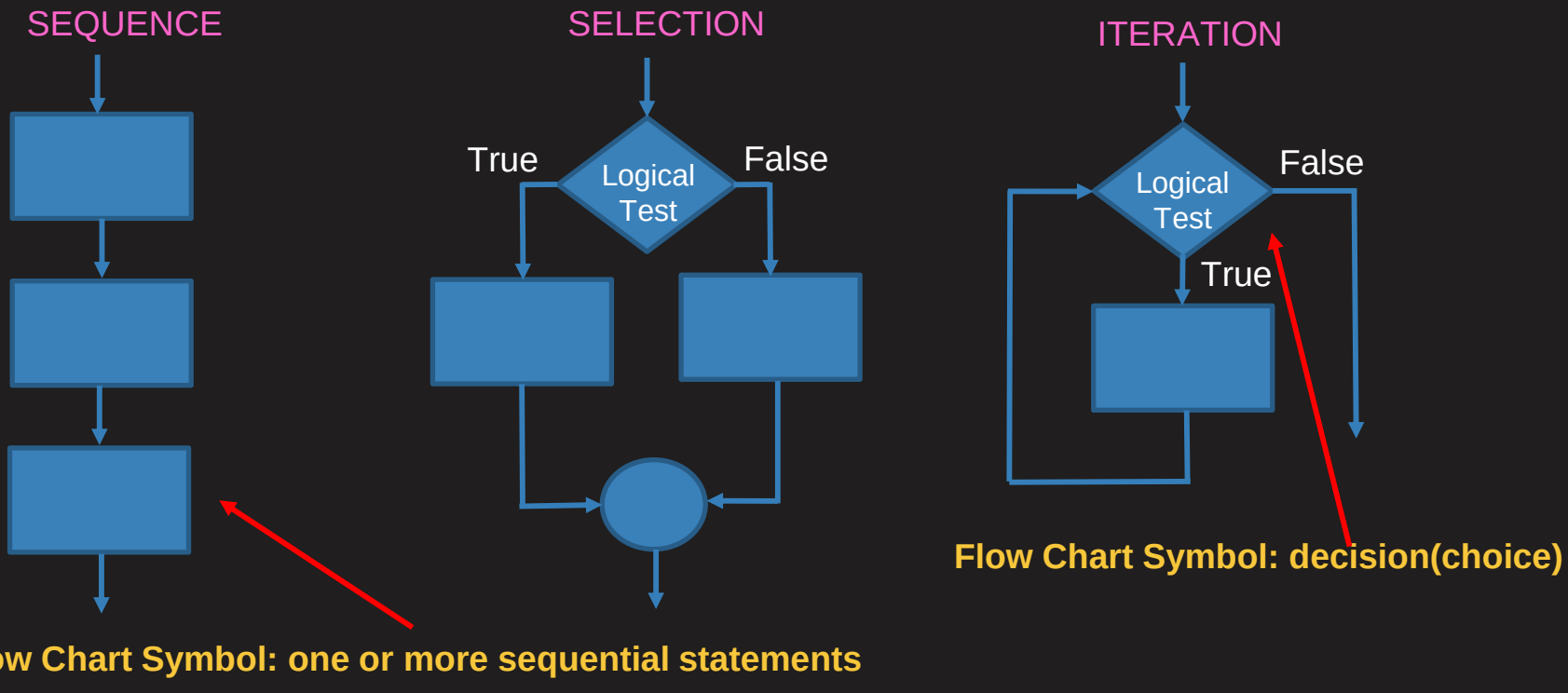
JAVA syntax to print “Hello World”

Try this: Click Compile, then execute



Program Flow Control Structures

There are three program flow control structures in all programming languages: SEQUENCE, SELECTION, ITERATION



Program Flow Control Structures

Sequence: Assignment or Function call statements

Assignment statements or calls to functions

Variable “a” is a memory location that can be changed. In this case “6” will be stored.

Assignment statement:

```
a = 2 + 4; // reads: put 2+4 which is 6 into the variable a
```

“//” means the following is a comment

“;” is placed at the end of each statement to tell the compiler you are done with the statement

Assignment operator

Math Operators
+ Add
- Subtract
*** Multiply**
/ Divide

Variables and Statements

Try this: enter code, compile, execute

int varA; // reserve 4 bytes varA

Compile | Execute | Share Project HelloWorld.java ×

```
1  public class HelloWorld {
2      public static void main(String []args){
3          int varA; // reserve a memory location for the variable varA
4          varA = 10; // assign 10 to variable varA
5          varA = varA + 5; // add 5 to varA
6          System.out.println(varA);  // print to the console box varA and then a carriage return
7          System.out.println("Hello World"); // print to the console box Hello World then a carriage return
8      }
9  }
```

Console:
What do you expect?
What do you see?

Short cuts:
int varA = 10; // you can define type, variable and assign value on one line
varA += 5; // add 5 to varA-this is the same as line 5

Variables and Statement Practice

No “MAGIC NUMBERS” – a=3, b=7, c=8, d=12, e=2.0; save your program for each

1 add a,b,c print result
2 add b,c and divide by c, print result
3 add e,a, print result
4 use shortcut method of math to add c to varable count and print count

Program Flow Control Structures

Selection (conditional statements IF-THEN-ELSE)

IF

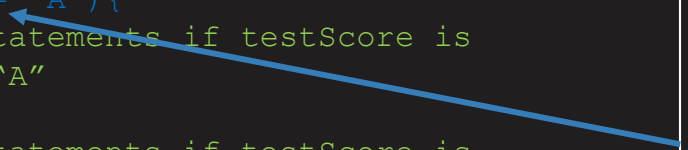
```
If (a>b){
     // execute statements if a is
     // greater than b
 }
```

IF-ELSE-IF

```
If  (speed <= 50){
    // execute statements if speed is
    // less than or equal to 50
} else if (speed < 10){
    // execute statements if speed is
    // less than 10
}
```

IF-ELSE

```
If  (testScore == "A"){
    // execute statements if testScore is
    // equal to "A"
} else {
    // execute statements if testScore is
    // not equal to "A"
}
```



Relational Operators:

- == equal
- != not equal
- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to
- && AND
- || OR
- ! NOT

“If-Then-Else” Conditional Statement

JAVA syntax for a “Constant” value that does not change

Comment has variable “units”

Try this: enter code, compile, execute

Compile | Execute | Share Project HelloWorld.java ×

```
1  public class HelloWorld{
2      public static void main(String []args){
3
4          // initialize variables
5          int carSpeedValue = 40; // car speed in MPH
6          final int SPEEDING = 55; // constant- start of speeding
7          final int SLOW_SPEED = 20; // constant-start of going too slow
8
9          // Check a car speed for speeding or going too slow
10         if (carSpeedValue >= SPEEDING){
11             System.out.println("Car is speeding");
12         } else if (carSpeedValue <= SLOW_SPEED){
13             System.out.println("Car is going too slow");
14         } else {
15             System.out.println("Car is moving within the speed limits");
16         }
17
18     }
```

Console:
What do you expect?
What do you see?

Change carSpeedValue to 65
Compile, execute
Change carSpeedValue to 15
Compile, execute

Comment for this section of code (code “Why”)

“If-Then-Else” Practice

1 Set isMoving true, if moving and at a speed x, decrease speed by 10%, print speed
2 Set isMoving false, if moving apply bike brakes and print “Appling Bike Brakes”, or if stopped print “Bike Already Stopped”; change isMoving to true
3 If variable grade is >=90 print A, >=80 print B, >=70 print C

Program Flow Control Structures

A “Switch” structure is used:

- To improve code readability by eliminating if-then-else statements
- It will jump to a code section based on a discrete value(e.g. Mon, Tue, Wed, Thur, Fri, Sat, Sun)

Selection (switch statement)

```
Switch ( value )
{
    case value1:
          // execute statements if value1 is equal to value
          break; // When statements are completed exit switch construct
    case value2:
         // execute statements if value2 is equal to value
         break;
    case value3:
        // execute statements if value3 is equal to value
        break;
    default:
        // default is optional – runs if there is no value that matches value1-3
}
```

“switch” Conditional Statement

Enter code, compile, execute

Console:
What do you expect?
What do you see?

Change motionState to 1
Compile, execute
Change motionState to 2
Compile, execute
Change motionState to 3
Compile, execute

Compile | Execute | Share Project HelloWorld.java ×

```
1  public class HelloWorld{
2      public static void main(String []args){
3          int motionState   = 2; // present state
4          final int STOP    = 0; // stop state
5          final int RUNNING = 1; // running state
6          final int BRAKING = 2; // braking state
7
8          switch (motionState){
9              case STOP:
10                 System.out.println("Stopped state");
11                 break;
12             case RUNNING:
13                 System.out.println("Running state");
14                 break;
15             case BRAKING:
16                 System.out.println("Braking state");
17                 break;
18             default:
19                 System.out.println("default state");
20         }
21     }
22 }
```

Typically an if statement on some condition here would change the motionState variable to RUNNING(1) or BRAKING(2)

“Selection” Practice

1 Month is a variable that has a value for a month (1-12); based on Month variable print out month name

Program Flow Control Structures

Looping (Conditional)

While-Do:

While (comparison statement)
{
    // statements will execute until while
    // comparison is true
}

```
For example:

int x = 1; //initialize x

while(x < 20){
    System.out.print(x);
    x = x + 1;
}
```

Looping (Repetition)

For loop:

for (init counter; comparison statement; increment count)
{
    // will execute Statements until comparison
    // statement is true (N times)
}

```
For example:

for (int x = 10; x < 20; x = x+1){
    System.out.print("value of x : " + x );
    System.out.print("\n");
}
```

“While – do” Conditional Loop

Enter code, compile, execute

Compile | Execute | Share Project HelloWorld.java ×

```
1  public class HelloWorld {
2      public static void main(String []args){
3          int i = 5;          // i,j,k typical variables for counters
4          int LOOP_END = 0; // loop count
5
6          while (i > LOOP_END)
7          {
8              System.out.println("Hello World");
9              i--; // decrement i short cut
10         }
11     }
12 }
```

Console:
What do you expect?
What do you see?

Short cut to decrement a variable-same as (i = i – 1)

Use “CONSTANT” name – NO MAGIC NUMBERS

How many were printed?

Magic Number – a number in a program with no reason for its value

“While-Do” Practice

1 While a count value is less than 11 increase count by one and print count value

“For” Repetitive Loop

Try this: enter code, compile, execute

Compile | Execute | Share Project HelloWorld.java ×

```
1  public class HelloWorld
2  {
3      public static void main(String []args)
4      {
5          int i;              // i,j,k typical variables for counters
6          int LOOP_END = 5; // loop count
7
8          for (i = 1; i < LOOP_END; i++)
9          {
10             System.out.println(i);
11             System.out.println("Hello World");
12         }
13     }
14 }
```

Console:
What do you expect?
What do you see?

Use white space for readability

How many were printed?

Short cut to increment a variable-same as i = i +1

“For” Practice

1 For a count variable that is 10 increase the count by 2 until it reaches 20 and print out each increased count value

Program Function

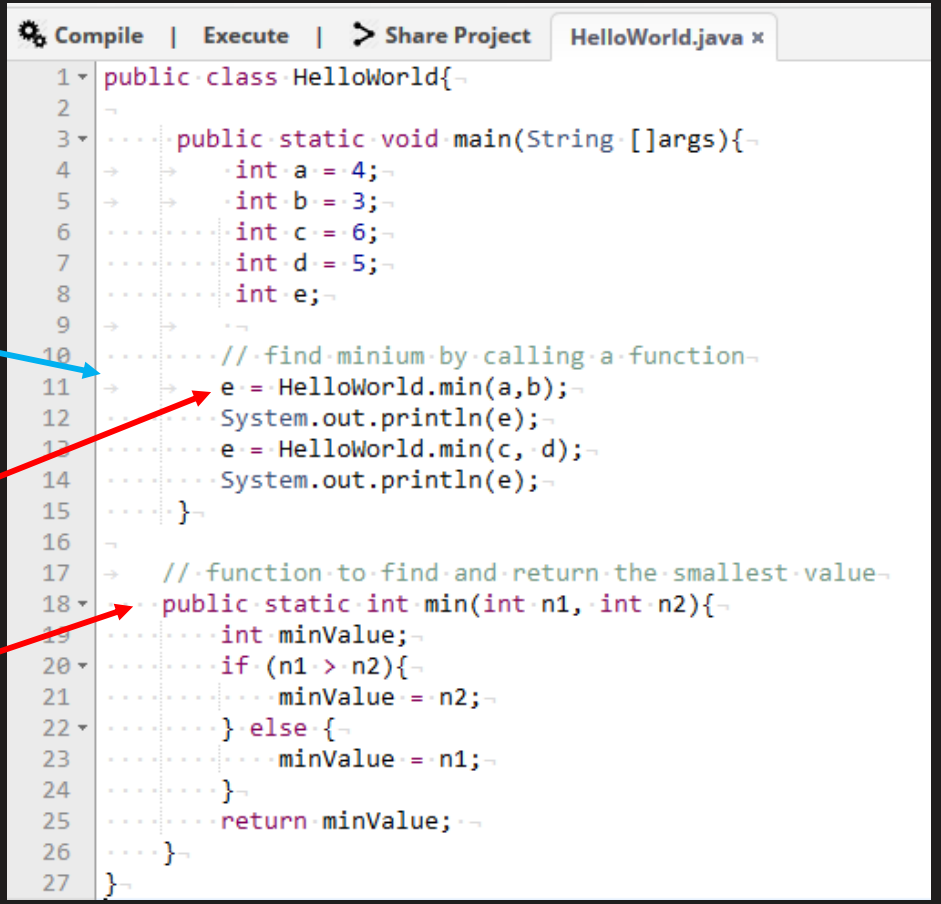
If a sequence of code is repeated several times, it more efficient to “call” a function

Try this: enter code, compile, execute

Console:
What do you expect?
What do you see?

Call function “min”

JAVA function syntax



Program Style Best Practices

- READ the software style guide in the software handbook!!!!
- NO MAGIC NUMBERS (numbers that have no explanation of what they are) – use constant names – this helps in readability and changing a name value will change it in all instances
- Use parentheses(“(…)”) around comparisons
- Use white space in math operations(e.g. a = 2 + 3; not a=2+3;)
- Comment every variable and note the “units”
- For code use a comment on a separate line with a line space before the comment
- Use line space between sections of your code

Team Program Style Guide

- To provide a consistent view of a program for others to read, every language has a “Style Guide”

- Refer to the Team 2228 JAVA style guide in the “Team 2228 Software Handbook”

GOLDEN RULE OF PROGRAMMING:

COMMENT!!-COMMENT!!-COMMENT!!

Programming Misconception: “Software is self documenting”. **IT IS NOT**

- Program headers and **“COMMENTS”** provide the **“WHY”**
- **T**he code provides the **“HOW”.**

Part II Objectives

- Understand “Class” and “Object”
- Understand State Machines

How Do You See The World

- As humans we categorize “objects” that have similar “attributes” and “functions” with a group “class” name

  For example: cars, bikes, airplanes, houses, dogs, cats, etc

- We categorize to reduce complexity

Why would we want to do this in software? – Improved reliability:

- Minimizes effects on program changes
- Provides a standard structure for object data and functions (this is called “encapsulation”)
- Provides a structure to reuse code

Categorization - Class

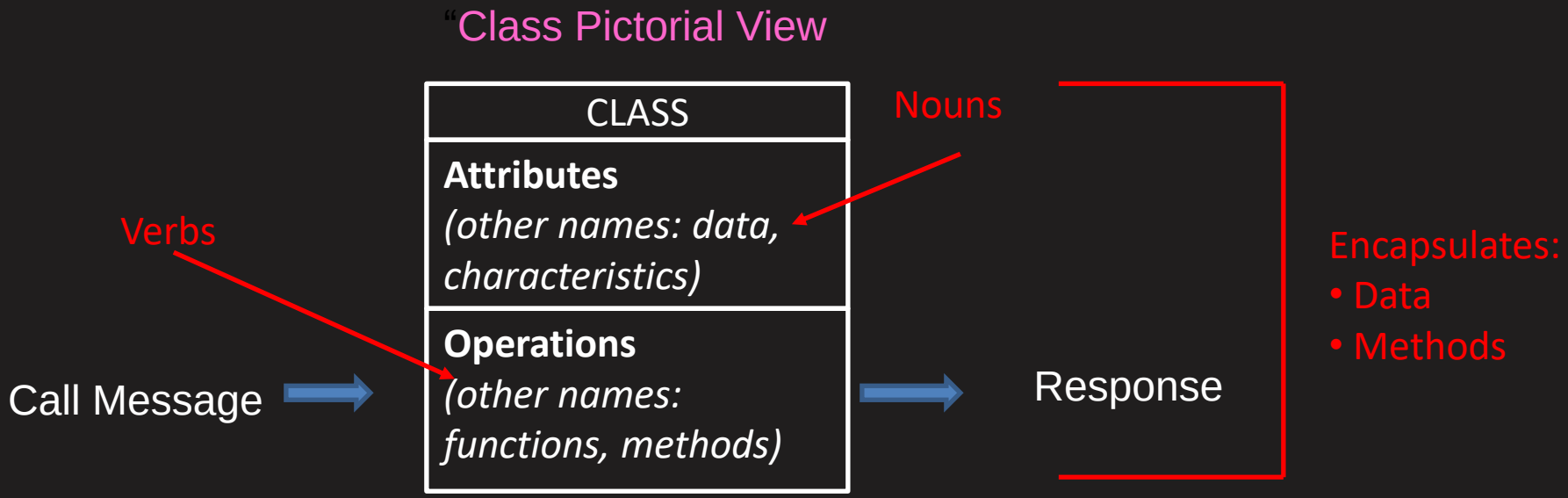
- A “Class” is a bundle of software of related data and behaviors(“methods”) that models real world objects. It is the blueprint or prototype from which program “Objects” are created

The structure of a class is as follows:

- Variable (“field”) definitions
- Constructor definition (How a program constructs objects with initial conditions)
- Method definitions (functions of the object)

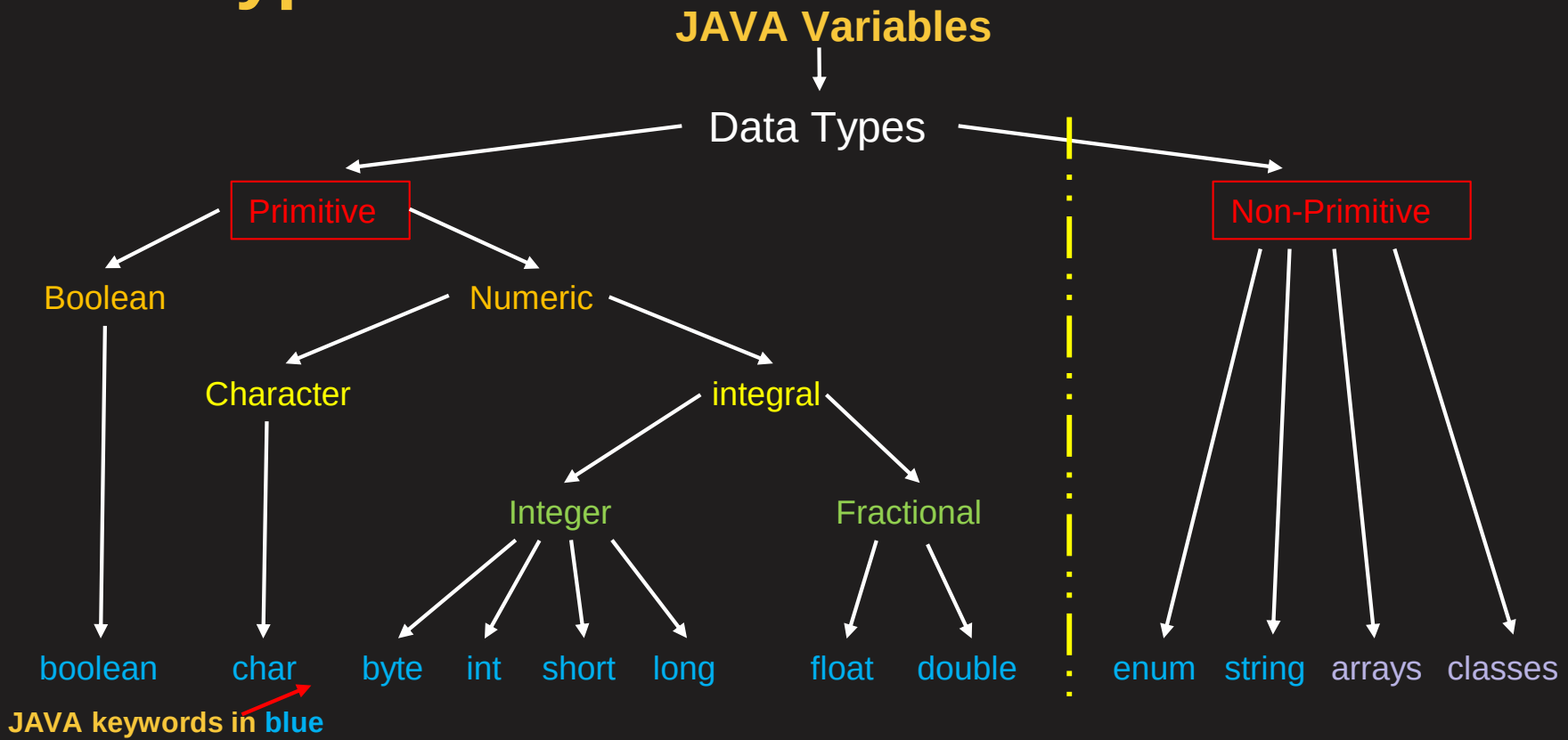
Class Pictorial View (UML)

A “CLASS” is a template / blue print for a group of common objects (e.g. bike)



UML – Universal Modeling Language

Data Types Revisited



Keywords are reserved words that have a predefined meaning in the JAVA syntax. They are used by the complier to create machine code structures

Primitive Data Types

Type	Contains	Default	Size	Range
`boolean`	`true` or `false`	`false`	1 bit	NA
`char`	Unicode character	`\u0000`	16 bits	`\u0000` to `\uFFFF`
`byte`	Signed integer	0	8 bits	-128 to 127
`short`	Signed integer	0	16 bits	-32768 to 32767
`int`	Signed integer	0	32 bits	-2147483648 to 2147483647
`long`	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
`float`	IEEE 754 floating point	0.0	32 bits	1.4E-45 to 3.4028235E+38
`double`	IEEE 754 floating point	0.0	64 bits	4.9E-324 to 1.79769313486231570E+308

Class Data Type and Object Construction

A “Class” is a composite “Type” that contains many values. This is different from a primitive type that contains only one value.

JAVA Syntax

```
import Car;

Car _ford;

_ford = new Car(string “blue”, string “gas”);
```

Point to where the program will find the “Class” Car (Car.java)

Create a variable that can point to an object

“ford” specific attributes

“_ford” is now a specific “Instance” of the “Class” Car

Keyword that loads the blue print code, assigns the object pointer name, and initializes attributes (if needed)

Keywords are reserved words that have a predefined meaning in the JAVA syntax. They are used by the complier to create machine code structures

Calling Methods

We talk to object methods via calls

```
Public class driving{

    Import Car
    Public static void main(string []args){
        // declare variables
        final int DRIVE  = 1;  //gear status-Park,drive,reverse
        int carSpeedStpt = 50; //mph
        Car _ford;              //object ford

        // construct object "ford"
        ford = new Car(string "blue", string "gas");

        //To drive the ford you need to "TURN ON" the engine and put the gear
        _ford.turnOn();
        _ford.setGearStatus(DRIVE);

        //set the car speed goal
        _ford.setSpeed(carSpeedStpt);
    }
}
```

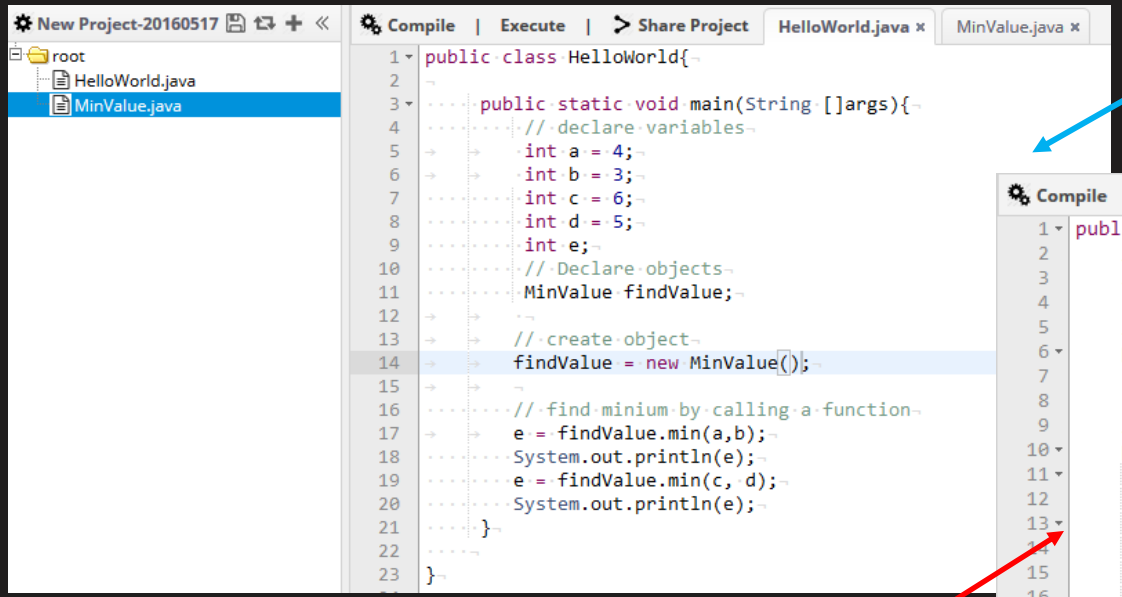
Class "Car"

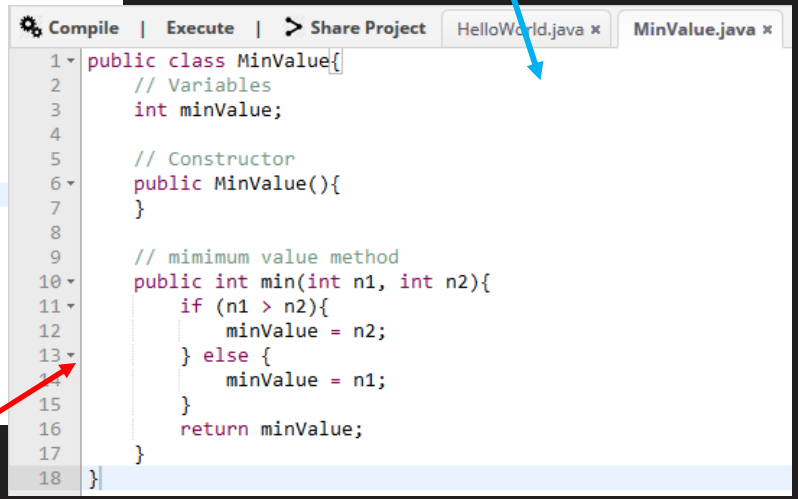
Car.java
carColor carFuel carSpeedStpt gearStatus
turnOn setGearStatus setSpeed

Call method in object "ford"

JAVA syntax for a call
Object name.object method(method parameters)

Class example





Try this: enter code, compile, execute

Console:
What do you expect?
What do you see?

Now MinValue Class could be used in other programs, however, there are no comments-Shame-Shame – see Software Handbook on how to comment classes

Method Names

Typical class method names:

Methods name to access variables:
“set”variable name (e.g. setSpeedStpt, setDrivingStatus)
“get”variable name (e.g. getCarSpeedValue, getDrivingStatus)
Also, if you want a specific status: use isDrivingStatus
(e.g. isDrivingStatus(DRIVING), isUp, IsDown)

For methods:
- Method names typically are verbs
- To improve readability think about how a call statement would read
(i.e. object.method(parameters)

JAVA Modifiers

- Access Modifiers are used to tell the compiler the accessibility of a class, its data and methods with respective to other classes. We will look at only “public” and “private”
  - Public: Declaring a class, data, or methods public then other classes have access to theses members where the class is visible.
  - Private: Declaring data or a method private restricts its visibility to the class that it is in. Private members are not accessible from any other class within a class’s package also.
- Other modifiers: “Static” and “final”
  - Static: Static modifier for data tells the complier that this variable is a class variable. Changing its value will change it in all objects of this class.
  - Final: Final modifier tells the compiler that this data value cannot be changed after its initialization. It is a constant.

For example:

```
public Class Test{ // standard class syntax
    public int var1= 20; // everyone can see this variable
    final int VAR3 = 25; // This is a constant value in Test class
    public void testMethod(){ // standard class method – “void”: return no value
    }
}
```

Name Conventions in Robotics

To improve readability the following conventions should be used:

- Abbreviations should be a minimum of 3 characters!
- Object names should be in directory style
- (e.g. ModuleName-ModuleComponent/Adjective-ComponentType
  i.e. gatherRollerMotor, driveTrainLeftWheelMotor, elevatorboulderPresentSensor, elevatorRaisedLimitSwitch)

- Variable name suffixes:
  For Boolean or mode type variables: ”Status” (e.g. elevatorUpPostionStatus)
  For variables that are constants that can be changed: “Stpt” (setpoint)
  (e.g. speedStpt)
  For variables that analog signals: “Value” (e.g. elevatorAngleValue)
- Typical component states:
  Cylinder: EXTENDED/RETRACTED
  Motor: FORWARD/REVERSE and ZERO_SPEED/AT_SPEED
  Switch: CLOSED/OPEN or ON/OFF

Other Non-Primitive Data Types

- Three other data types are built into the JAVA language: String, enum-eration, and arrays
- Constructing these data types:
- String:

```
String HELLO_MSG = “Hello World”
```

- Array:

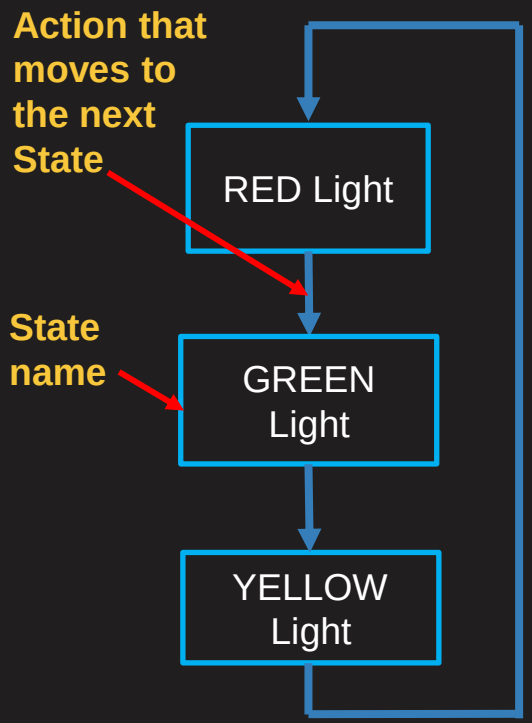
```
// declare an array of intergers
Int anArray;
// allocate memory for the integers
anArray[] =  new int[10]; // element locations are 0-9
// shortcut to create and initialize an array
Int[] anArray = {0,1,2,3,4,5,6,7,8,9}
```

- Enum: Is a set of predefined constants

```
import java.util.Enumeration;
public enum Level {
    LOW,        // value 0
    MEDIUM,   // value 1
    HIGH         // value 2
}
// create object
Level levelState;
```

State Machines

State Diagram




```
Enum  Light {RED, GREEN, YELLOW}
// make light state object
Light lightState;
// initialize state machine, start timer(not shown), set timer status
lightState = RED;
redTimerStatus = true;
while(true){     // do forever
    Switch (lightState){
        case RED:
            if (redTimerStatus == false){
              // start GREEN light timer(not shown), set timer status, change state
               lightState = GREEN;
            }
            break;
        case GREEN:
            if (greenTimerStatus == false){
              // start YELLOW light timer(not shown), set timer status, change state
               lightState = YELLOW;
            }
            break;
        case YELLOW:
            if (yellowTimerStatus == false){
               // start RED light timer(not shown), set timer status, change state
               lightState = RED;
            }
            break;
    }
}
```