The increment operator

- ++ adds 1 to a variable
  - It can be used as a statement by itself, or within an expression
  - It can be put before or after a variable
  - If before a variable (pre-increment), it means to add one to the variable, then use the result
  - If put after a variable (post-increment), it means to use the current value of the variable, then add one to the variable
- The same applied to decrement operator

Examples of ++

```java
int a = 5;
a++;
// a is now 6

int b = 5;
++b;
// b is now 6

int c = 5;
int d = ++c;
// c is 6, d is 6

int e = 5;
int f = ++e;
// e is 6, f is 5

int x = 10;
int y = 100;
int z = ++x + y++;
// x is 11, y is 101, z is 111
```

Confusing code is bad code, so this is very poor style

char

- The primitive type char
  - Just stored as numbers
  - Each char as a unique integer value (based on Unicode standard)
- You can use characters in arithmetic (they will automatically be converted to int)
  > char ch = 'A';
  > ch + 1
  66
  > char ch2 = (char) (ch + 1) // cast result back to char B

The primitive type char
- Just stored as numbers
- Each char as a unique integer value (based on Unicode standard)
- You can use characters in arithmetic (they will automatically be converted to int)
  > char ch = 'A';
  > ch + 1
  66
  > char ch2 = (char) (ch + 1) // cast result back to char B
Casting with Numeric PrimitiveTypes

- Numeric types are considered wider or narrower than other numeric types
  - Based how much memory space they occupy
- Java doesn’t mind if you assign a narrow value to a wide variable: `double n = 3;`
- Java is not happy if you assign a wide value to a narrow variable: `int n = 3.5; //illegal`
- But if you want to narrow (assign a wider type to a narrower type), you have to cast it:
  - `double d = 3.5;`
  - `i = (int) d; //legal due to casting`
- Java checks to make sure that the cast works, and gives you an error if it didn’t

Casts with Reference Types

```java
public class Vehicle{
    protected int regNum;
    public Vehicle(int r){
        regNum = r;
    }
    public int getRegNum(){
        return regNum;
    }
}
```

```java
public class Car extends Vehicle{
    protected int numDoors;
    public Car(int r, int n){
        super(r); //or regNum = r
        numDoors = n;
    }
    public int getDoors(){
        return numDoors;
    }
}
```

Casts with Reference Types (contd..)

- `Vehicle v = new Car(45,4); //valid`
- `v.getRegNum()`
  - 45
- `v.getDoors()`
  - Error
- Illegal because “v” could potentially refer to other types of vehicles that are not cars
- The solution here is to use type-casting.

Casts with Reference Types (contd..)

- If, for some reason, you happen to know that “v” does in fact refer to a Car, you can use the `type cast`
  - Use `instanceof` keyword to find that out
- Do `(Car)v` to tell the computer to treat “v” as if it were actually of type Car. So, you could do: `((Car)v).getDoors()`
Main

public static void main (String [] args)
- Must have the exact signature
  - Only variation allowed is name of the input parameter
- So main starts everything, how do we call main and provide inputs?

- To run a program recall
  - Command: java ClassName
    - This what calls the main method if the class has one
  - So we could pass arguments as follows:
    java ClassName list-of-arguments

Main with arguments example

public class ExampleArgs{
    public static void main(String [] args){
        System.out.println("Demo for Inputs args");
        for(int i = 0; i < args.length; i++){
            System.out.println(args[i]);
        }
    }
}

> java ExampleArgs ESE 112
Demo for Inputs args ESE 112

Note: Code works even if no arguments are passed to main() because JVM passes to main() a zero-length array of Strings and not a null

Example from DSP Lab

public class MP3Player {
    public static void main(String[] args) {
        String filename = args[0];
        StdPlayer.open(filename);

        while (!StdPlayer.isEmpty()) {
            Wave w = new Wave(StdPlayer.getLeftChannel(),
                                StdPlayer.getRightChannel());
            w.play();
        }

        StdPlayer.close();
        System.exit(0);
    }
}

Memory Management

- Memory is not infinite
- Stacks grow and shrink
- Heap
  - Grow when you dynamically allocate memory i.e. new Object()
  - If you do not manage the allocations then you will run out of this memory
    - Objects that will never be accessed or mutated again by application need to be reclaimed
      - This is known as Garbage Collection
- Some Languages like C/C++ leave it up to the programmer to do explicit memory management
- Java does automatic garbage collection
  - Done by JVM (Java Virtual Machine)
Recap Static vs. Dynamic

public class JustAdd {
    public int x;
    public int y;
    public int z;

    public static void main(String args[]) {
        x = 5;
        y = 10;
        z = x + y;
        System.out.println(z);
    }
}

Solution 1: If non-OOP is intended

public class JustAdd {
    public static int x;
    public static int y;
    public static int z;

    public static void main(String args[]) {
        x = 5;
        y = 10;
        z = x + y;
        System.out.println(z);
    }
}

Solution 2: If OOP is intended

public class JustAdd {
    int x;
    int y;
    int z;
    //Method will executed by an object of Type JustAdd
    public int sumZ() {
        x = 5;
        y = 10;
        z = x + y;
        return z;
    }
    public static void main(String args[]) {
        JustAdd myAdd = new JustAdd(); //Main must create object first
        System.out.println(myAdd.sumZ());
    }
}

Summary of Static vs. Non Static

- Its better to write main method in a separate class so you do not get confused

- Major Points
  - Static variables and methods belong to class
    - To access them in another file we do filename.methodname() or filename.variablename()
  - Variables and methods not declared static automatic become OO
    - Then first we must create the object
Summary contd..

- Static methods besides main method can take in parameters of other object types
  - This allows static method to access to particular object of interest
  - Example:

```java
public static void compareRadius(Circle c1, Circle c2){
    if(c1.getRadius() >= c2.getRadius()){  
        System.out.println("Circle 1 is greater");
    }
}
```