

Introduction to Programming

with Java, for Beginners

- ++ & -- operator
- Switch statement
- Passing arguments to Main
- Number Representation & Storage

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

ESE112

2

Examples of ++

```
int a = 5;
> a++
5
> a // a is now 6
6
int b = 5;
> ++b
6
> b
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

ESE112

3

The *decrement* operator

- -- subtracts 1 from a variable
 - Used similarly as ++ operator
- ```
>int a = 5;
>a-- //a is now 4
5
>--a // a is now 3
>3
>a
3

int c = 5;
int d = --c; // c is 4,
//d is 4

int x = 10;
int y = 100;
int z = --x + y--;
// x is 9, y is 99, z is 109
```

ESE112

4

## Syntax of the *switch* statement

- The syntax is:

```
switch (expression) {
 case value1 :
 statements ;
 break ;
 case value2 :
 statements ;
 break ;
 ...(more cases)...
 default :
 statements ;
 break ;
}
```
- The *expression* must yield an integer or a character
- Each *value* must be a literal integer or character
- Notice that colons ( : ) are used as well as semicolons
- The last statement in every case should be a *break*;
  - If missing, fall through the next case
- The *default*: case handles every value not otherwise handled

ESE112

5

## Example switch statement

```
switch (cardValue) {
 case 1:
 System.out.print("Ace");
 break;
 case 11:
 System.out.print("Jack");
 break;
 case 12:
 System.out.print("Queen");
 break;
 case 13:
 System.out.print("King");
 break;
 default:
 System.out.print(cardValue);
 break;
}
```

ESE112

6

## 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*

ESE112

7

## Main Args 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

ESE112

8

## What does the Computer Understand?

- At the lowest level, a computer has electronic “plumbing”
  - Operates by controlling the flow of electrons through very fast tiny electronic devices called transistors
- The devices react to presence or absence of voltage
  - Could react actual voltages but designing electronics then becomes complex
- Symbolically we represent
  1. Presence of voltage as “1”
  2. Absence of voltage as “0”

ESE112

9

## What does the Computer process & store?

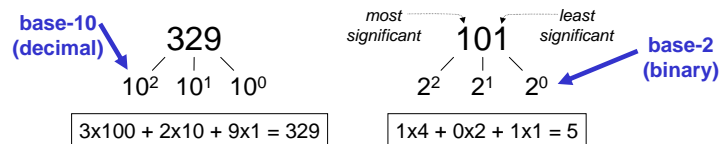
- An electronic device can represent uniquely only one of two things
  - Each “0” and Each “1” is referred to as a **Binary Digit** or **Bit**
  - Fundamental unit of information storage
- To represent more things we need more bits
  - E.g. 2 bits can represent four unique things: 00, 01, 10, 11
  - k bits can distinguish  $2^k$  distinct items
- Combination binary bits together can represent some information or data.
  - E.g. 00101001 can be Decimal value 65
  - Interpretation of binary string is what distinguishes data

ESE112

10

## Unsigned Integers

- Like decimal numbers: “329”
- “3” is worth 300, because of its position, while “9” is only worth 9



ESE112

11

## Unsigned Integers (cont.)

- An  $n$ -bit unsigned integer represents  $2^n$  values
  - From 0 to  $2^n-1$

| $2^2$ | $2^1$ | $2^0$ | val |
|-------|-------|-------|-----|
| 0     | 0     | 0     | 0   |
| 0     | 0     | 1     | 1   |
| 0     | 1     | 0     | 2   |
| 0     | 1     | 1     | 3   |
| 1     | 0     | 0     | 4   |
| 1     | 0     | 1     | 5   |
| 1     | 1     | 0     | 6   |
| 1     | 1     | 1     | 7   |

ESE112

12

## Hexadecimal Notation

- It is often convenient to write binary (base-2) numbers as hexadecimal (base-16) numbers instead
  - Fewer digits: four bits per hex digit

| Binary | Hex | Decimal | Binary | Hex | Decimal |
|--------|-----|---------|--------|-----|---------|
| 0000   | 0   | 0       | 1000   | 8   | 8       |
| 0001   | 1   | 1       | 1001   | 9   | 9       |
| 0010   | 2   | 2       | 1010   | A   | 10      |
| 0011   | 3   | 3       | 1011   | B   | 11      |
| 0100   | 4   | 4       | 1100   | C   | 12      |
| 0101   | 5   | 5       | 1101   | D   | 13      |
| 0110   | 6   | 6       | 1110   | E   | 14      |
| 0111   | 7   | 7       | 1111   | F   | 15      |

### Hex to Decimal

ESE112      $1A7 = 1 \times 16^2 + 10 \times 16^1 + 7 \times 16^0 = 423$  (base 10)

13

## char

- The primitive type `char`
  - Just stored as numbers
  - Each char as a unique unsigned integer value (character code)
    - Based on Unicode standard
    - Uses 16-bits to represent characters

- You can use characters in arithmetic (they will automatically be converted to `int`)

```
> char ch = 'A';
> ch + 1
66
> ch2 = 67
C
```

ESE112

14

## Two's Complement Signed Integers

- Range of an n-bit number:  $-2^{n-1}$  through  $2^{n-1} - 1$ 
  - Note: most negative number ( $-2^{n-1}$ ) has no positive counterpart

| $2^3$ | $2^2$ | $2^1$ | $2^0$ |   | $2^3$ | $2^2$ | $2^1$ | $2^0$ |    |
|-------|-------|-------|-------|---|-------|-------|-------|-------|----|
| 0     | 0     | 0     | 0     | 0 | 1     | 0     | 0     | 0     | -8 |
| 0     | 0     | 0     | 1     | 1 | 1     | 0     | 0     | 1     | -7 |
| 0     | 0     | 1     | 0     | 2 | 1     | 0     | 1     | 0     | -6 |
| 0     | 0     | 1     | 1     | 3 | 1     | 0     | 1     | 1     | -5 |
| 0     | 1     | 0     | 0     | 4 | 1     | 1     | 0     | 0     | -4 |
| 0     | 1     | 0     | 1     | 5 | 1     | 1     | 0     | 1     | -3 |
| 0     | 1     | 1     | 0     | 6 | 1     | 1     | 1     | 0     | -2 |
| 0     | 1     | 1     | 1     | 7 | 1     | 1     | 1     | 1     | -1 |

ESE112

15

## Bits to Bytes

- Often storage is described in terms of bytes
  - 1 Byte = 8 bits
  - 16 bits = 2 Bytes (char in Java)
  - 32 bits = 4 Bytes (int in Java)
  - 64 bits = 8 Bytes (double in Java)
  - 1024 bits = 1 Kilo-Byte (KB)
  - 1024 x 1024 bits = 1 Mega-Byte (MB)
  - 1024 x 1024 x 1024 = 1 Giga-Byte (GB)

ESE112

16

## More on Storage

- Stack: store temporary data within method or procedure
  - E.g. local variables within a method or variable declared in a for loop
    - After the method/block exits, the variables are no longer available
- Heap: remember data that can be accessed outside of methods and blocks
  - E.g. Objects are stored on heap
    - If you declare an array inside a method, you can access its contents even after you exit the method

ESE112

17

## 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)

ESE112

18