

# CIS 27P

Java for Programmers

Class 3

# Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement
+=	Operator Assignments
-=	
*=	
/=	
%=	

# Bitwise Operators

$\sim$	Bitwise unary NOT
$\&$	Bitwise AND
$ $	Bitwise OR
$\wedge$	Bitwise exclusive OR
$\gg$	Shift right
$\ggg$	Shift right zero fill
$\ll$	Shift left
$\&=$	Bitwise & operator

# Relational Operators

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

# Logical Operators

&	Logical AND
	Logical OR
^	Logical XOR
	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
==	Equal to
!=	Not equal to
?=	Ternary if-then-else

# Flow Control

- If statement
  - Nested if's
  - else if
  - block vs. single statement
- Switch statement
  - byte, short, int, char
  - more efficient
  - default optional

# Loops

- `while() {}`
  - takes boolean
- `do {} while()`
  - evaluated after the loop
- `for ( init; condition; iteration ) {}`
  - multiple inits and iterations
  - declared variables only exist in loop
  - no need for init, condition, or iteration

# Jump Statements

- Considered sloppy by most
- `break`
  - exits loop
  - does not work when inside switch
- `break label`
  - usually in nested loops
  - jumps to block marked with label
- `continue`
  - skips to next iteration of loop

# Miscellaneous

- `(char)System.in.read();`
  - reads a character from the keyboard
  - throws `java.io.IOException`

# Exercise

Write a program that executes an endless loop. Inside the loop, ask the user to input a digit. Convert the digit into a byte and display the value of each bit of the byte. If the user enters 0, the program should exit.

Sample Output:

```
Enter a digit:
```

```
9
```

```
The bit pattern is:
```

```
0 0 0 0 1 0 0 1
```

```
Enter a digit:
```

# Classes

- Class: template for an object
- Object: an instance of a class
- File name requirements
- Instance Variables
  - declaration
  - usage
- Methods
  - declaration
  - parameters
- Constructors
- Object Usage
- Garbage collection
- this keyword
- finalize()

```
// Employee.java
class Employee
{
    int id, salary; // instance variable declaration
    String title;

    // constructor naming convention, no return val
    Employee( int id, int salary, String title )
    {
        this.id = id; // hiding
        this.salary = salary; // this keyword
        this.title = title;
    }

    // return value
    int promote( int amount, String newTitle )
    {
        salary += amount;
        this.title = newTitle;
        return salary;
    }

    protected void finalize()
    {
        id = salary = 0;
        title = "";
    }
}
```

```
class UseEmployee
// obj usage, garbage collector
{
    public static main( String args[] )
    {
        Employee emp = new Employee(0, 30000, "Staff");
        int newSalary = emp.promote( 10000, "Manager");
        System.out.println("New Salary:"+newSalary);
    }
}
```