

# CS18000: Problem Solving and Object-Oriented Programming

Prof. H.E. Dunsmore

# Video 1

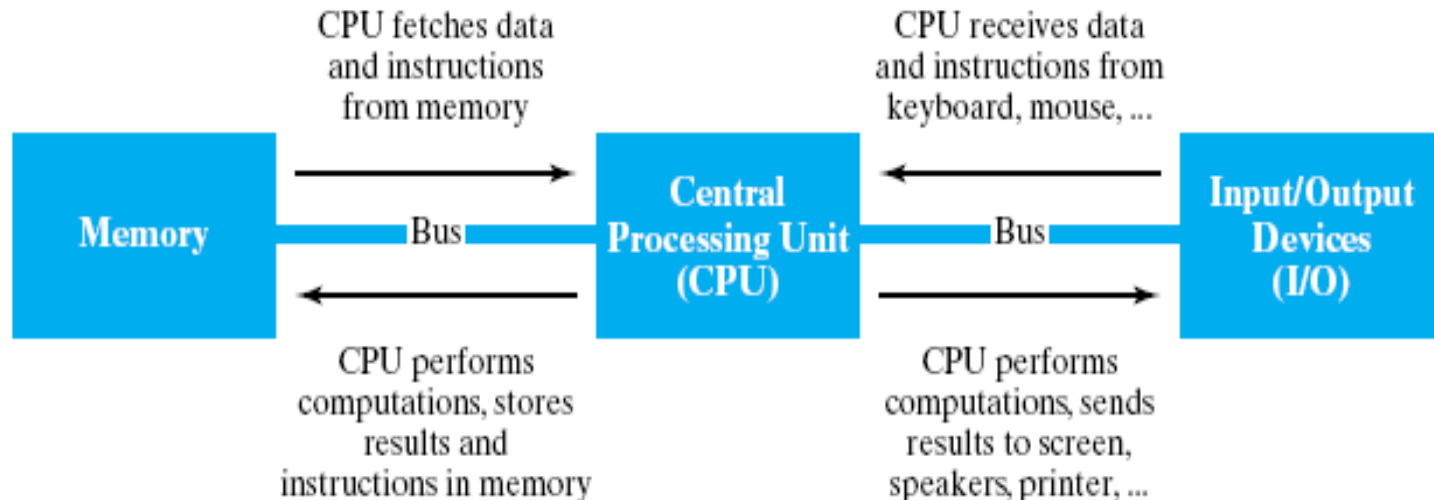
## Algorithms, Abstraction, and Number Systems

# Problem Solving

- Examples
  - Assemble a bookcase: directions
  - Bake a cake: recipe
- Algorithm:
  - Step-by-step *series of instructions* to solve a problem (must be correct, complete, must end)
- Abstraction:
  - The creation of a *concept* from specific *examples*

# Von Neumann Architecture

- Although specific components may vary, virtually all modern computers have the same underlying structure
  - known as the *von Neumann architecture*
  - named after computer pioneer, John von Neumann
- The von Neumann architecture identifies 3 essential components
  1. *Input/Output Devices (I/O)* allow the user to interact with the computer
  2. *Memory* stores information to be processed as well as programs (instructions specifying the steps necessary to complete specific tasks)
  3. *Central Processing Unit (CPU)* carries out the instructions to process information



# Memory and Storage Sizes

Unit	Size	Bytes	Practical Measure
byte	8 bits	$2^0 = 1 = 10^0$	A single character
kilobyte (KB)	1,024 bytes	$2^{10} = 1,024 \approx 10^3$	A paragraph of text
megabyte (MB)	1,024 kilobytes	$2^{20} = 1,048,576 \approx 10^6$	A minute of MP3 music
gigabyte (GB)	1,024 megabytes	$2^{30} = 1,073,741,824 \approx 10^9$	A half hour of video
terabyte (TB)	1,024 gigabytes	$2^{40} = 1,099,511,627,776 \approx 10^{12}$	80% of a human's memory capacity

# Number Systems

- Positional numbering
- Value of a digit is related to the distance of its position from the decimal point
- The system has a base, e.g., 2, 8, 10, or 16
- Each position multiplies or divides value by the base

# Examples

- In decimal:

$$352 = 3 \times 10^2 + 5 \times 10^1 + 2 \times 10^0$$

$$352 = 3 \times 100 + 5 \times 10 + 2$$

- In binary:

$$1110 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$1110 = 1 \times 8 + 1 \times 4 + 1 \times 2 = 14$$

- In hexadecimal:

$$3F = 3 \times 16^1 + F \times 16^0$$

$$3F = 3 \times 16 + 15 \times 1 = 48 + 15 = 63$$

# Conversions

- Algorithm:
  - Divide number by base
  - Prepend remainder to the result string
  - Replace number by quotient
  - Repeat until quotient is zero



# Example

- 42 to base 2:

$$42 / 2 = 21 + 0 \text{ remainder}$$

$$21 / 2 = 10 + 1 \text{ remainder}$$

$$10 / 2 = 5 + 0 \text{ remainder}$$

$$5 / 2 = 2 + 1 \text{ remainder}$$

$$2 / 2 = 1 + 0 \text{ remainder}$$

$$1 / 2 = 0 + 1 \text{ remainder}$$

Answer: 101010

# Video 2

## Storing Integers

# Finite Precision

- Computers store numbers using bits (base 2)
- Bits organized into bytes (8 bits)
- Bytes organized into words, typically
  - 4 bytes or
  - 8 bytes
- 4 bytes = 32 bits
- 8 bytes = 64 bits
- So, can only represent integers that “fit” into that many bits
- And, what to do about negative numbers?

# Interpreting Bits

3-bit Word	Unsigned Interpretation	Unsigned N-bit Generalization	Signed Interpretation	Signed N-bit Generalization
000	0	0	0	0
001	1		1	
010	2		2	
011	3		3	$2^{N-1} - 1$
100	4		-4	$-2^{N-1}$
101	5		-3	
110	6		-2	
111	7	$2^N - 1$	-1	-1

4-bit Word	Unsigned Interpretation	Unsigned N-bit Generalization	Signed Interpretation	Signed N-bit Generalization
0000	0	0	0	0
0001	1		1	
0010	2		2	
0011	3		3	
0100	4		4	
0101	5		5	
0110	6		6	
0111	7		7	$2^{N-1} - 1$
1000	8		-8	$-2^{N-1}$
1001	9		-7	
1010	10		-6	
1011	11		-5	
1100	12		-4	
1101	13		-3	
1110	14		-2	
1111	15	$2^N - 1$	-1	-1

# Sign Bit

- The left most bit of a word is the “sign bit”
  - 0 -> positive
  - 1 -> negative
- Representation is called “two’s complement”
- Positive numbers get leading 0s to word size
- Negative numbers:
  - Convert positive value to binary
  - Flip all bits (0 <-> 1)
  - Add one

# Example

- Convert -42 to binary using an 8-bit “word”
- $42 = 101010$
- Fill to 8 bits =  $00101010$
- Flip bits (one’s complement) =  $11010101$
- Add one (two’s complement) =  $11010110$
- Useful site: <http://planetcalc.com/747/>

# Useful Consequences (Summary)

- Largest (signed) positive number that can be stored in N bits is N-1 1s  
 $= 2^{N-1} - 1$   
Example for 8 bits: 01111111 = 127
- Largest negative number that can be stored in N bits is a 1 followed by N-1 0s  
 $= -2^{N-1}$   
Example for 8 bits: 10000000 = -128



# What Could Go Wrong?

- Overflow
  - Sum of two positive numbers is “too positive”
  - Example:  $127 + 1$  in eight bits
- Underflow
  - Difference of two numbers is “too negative”
  - Example:  $-128 - 1$  in eight bits
- Warning: Java ignores (quietly) overflow and underflow

# Video 3

## Wheel Class

# Wheel Class

- In our life we encounter a lot of things
- For example, in a parking lot there are a lot of things (objects) of the class Vehicle
- Each Vehicle object has attributes, such as color, make, model, license number, ....
- In the Java programming language, we can create a class that can be used to make lots of objects
- We can make a Wheel class with attributes radius (like 27.5 inches) and material (like rubber)
- Every Wheel object can have different attributes
- We can even create the operations (methods) we want to be able to perform on a Wheel object

# Wheel Class

```
public class Wheel {  
  
    // Wheel class has two attributes (instance variables)...  
  
    double radius;  
    String material;  
}
```

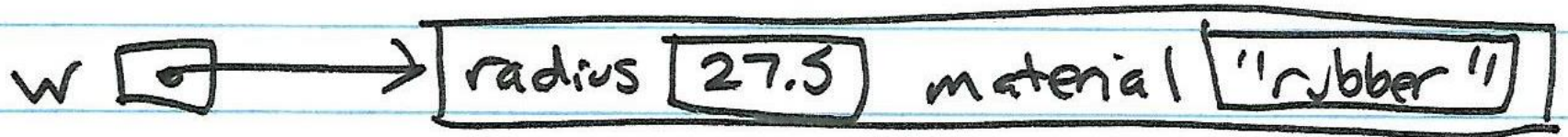
# Wheel Class

```
// Wheel class has two Constructors, one takes both a  
// radius and material  
// the other takes just a radius  
// "this.radius" means the radius of this object being  
// constructed  
// "radius" means the parameter value supplied
```

```
Wheel(double radius, String material) {  
    this.radius = radius;  
    this.material = material;  
}
```

# Wheel Class

```
Wheel w = new Wheel (27.5, "rubber");
```

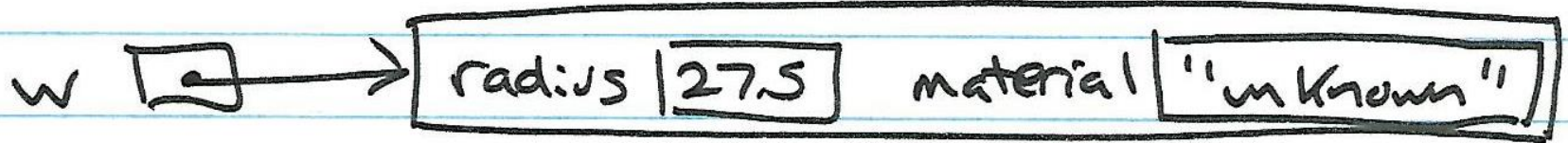


# Wheel Class

```
Wheel(double radius) {  
    this.radius = radius;  
    this.material = "unknown";  
}
```

# Wheel Class

```
Wheel w = new Wheel(27.5);
```





# Wheel Class

```
// Method names that begin with "get" are accessor  
// methods.  
// These provide a value of the return type -- either  
// double or String
```

```
double getCircumference() {  
    return 2 * Math.PI * radius;  
}
```

```
double getArea() {  
    return Math.PI * radius * radius;  
}
```

# Wheel Class

```
double getRadius() {  
    return radius;  
}
```

```
String getMaterial() {  
    return material;  
}
```

# Wheel Class

```
// Method names that begin with "set" are mutator  
// methods.  
// These store a new value for radius or material ... or  
// both.
```

```
void setRadius(double radius) {  
    this.radius = radius;  
}
```

```
void setMaterial(String material) {  
    this.material = material;  
}
```

# Wheel Class

```
void setRadiusAndMaterial(double radius, String material) {  
    this.radius = radius;  
    this.material = material;  
}
```

# Video 1

## Making a Class a Program

# Wheel Class

```
// A method that says "public static void main(String[]  
// args)"  
// is where execution will begin  
  
    public static void main(String[] args) {  
  
// area, circ, rad, and mat are local variables in the  
// main method  
    double area, circ, rad;  
    String mat;
```


# Wheel Class


```
// bicycle is a Wheel object with radius 27.5 and
// material "rubber"
    Wheel bicycle = new Wheel(27.5, "rubber");

// wagon is a Wheel object with radius 54.75 and material
// "unknown"
    Wheel wagon = new Wheel(54.75);

// change wagon's material from "unknown" to "wood"
    wagon.setMaterial("wood");
```

# Wheel Class

bicycle  radius | 27.5 | material | "rubber" |

wagon  radius | 54.75 | material | "unknown" |



# Wheel Class

```
// circ, area, and rad are the circumference, area, and
// radius of bicycle
    circ = bicycle.getCircumference();
    area = bicycle.getArea();
    rad = bicycle.getRadius();

// mat is the material of the wagon
    mat = wagon.getMaterial();

// bicycle radius changed from 27.5 to 32.0
    bicycle.setRadius(32.0);
```

# Wheel Class

```
// bicycle material changed from "rubber" to
// "polyethylene"
    bicycle.setMaterial("polyethylene");

// wagon radius and material changed
    wagon.setRadiusAndMaterial(88.35, "aluminum");

    }
}
```

# Methods and Headers

A method is one or more lines of code that can be executed for a class object by using the object name, method name, and any arguments...

```
wagon.setRadiusAndMaterial(88.35, "aluminum");
```

The first line of every method is its "header"

```
return-type name (parameters) {  
    ...  
}
```

# Methods and Headers

Some methods return a value (like double) and some return nothing (void)

Some methods have no parameters

Some have one, some have two, ....

Java chooses the method to run that matches the arguments when the method is called

```
wagon.setRadiusAndMaterial(88.35, "aluminum");
```

uses the setRadiusAndMaterial method that has a double and a String argument

# Methods and Headers

```
Wheel wagon = new Wheel(54.75);
```

uses the Constructor that has only a double argument

# Video 2

## Software Development

# Software Development Lifecycle

1. Understand the problem
2. Design a solution
3. Implement the solution
4. Test the solution
5. Maintenance

# How is Software Written?

- Programming language
  - high level language (complex statements)
  - assembly language (very simple instructions)
  - machine language (1s and 0s)
- Compilers and interpreters
- Integrated Development Environments (IDEs)
  - Allow editing of program files
  - Allow compiling, executing, and debugging programs



# How is Software Written?

## high level language

```
x=y+z;
```

## assembly language

```
load 1,y
```

```
load 2,z
```

```
add 1,2,3
```

```
store 3,x
```

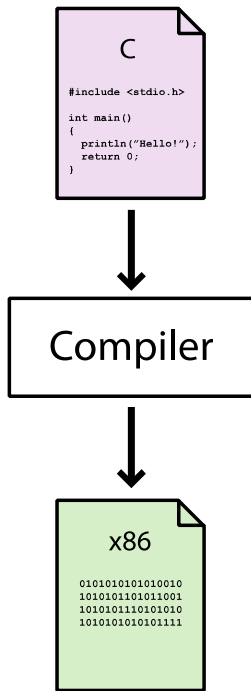
## machine language

```
10110 0001 101101
```

```
10110 0010 011100
```

# Program Translation

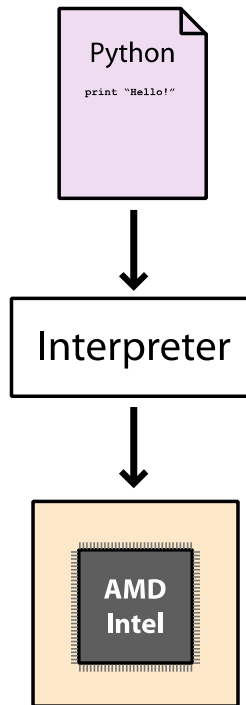
Program in high level language, such as C



Equivalent program in machine language

(a)

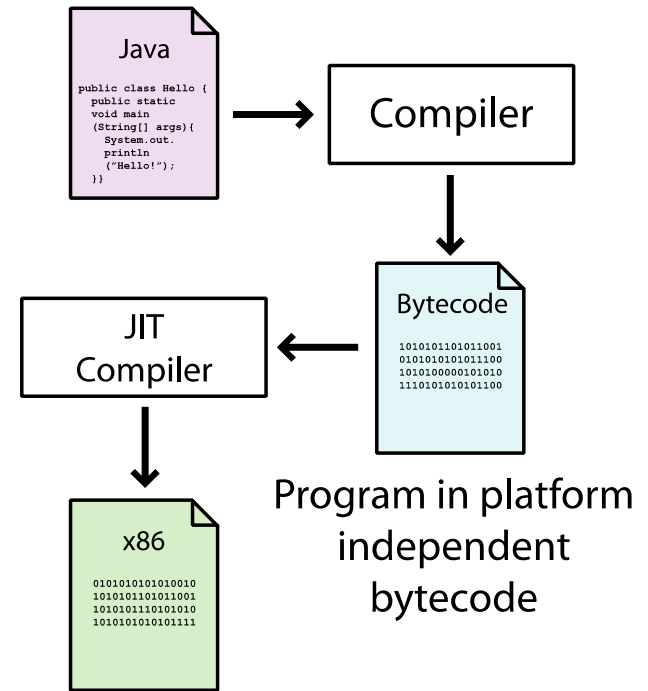
Program in high level language, such as Python



Executes program directly on CPU

(b)

Program in high level language, such as Java



Equivalent program in machine language

Program in platform independent bytecode

(c)

# Tools for Abstraction

- Object-Oriented Programming (OOP) is one way
- Class
  - Gives a name to the abstraction and creates a template for objects, which are instances of the abstraction
  - Examples: Wheel, Robot, Car, Chair
  - Includes variables and methods that define the details of the abstraction
- Variable
  - Names and stores a quantity
  - Examples: radius, name, speed, color
- Method
  - Names and defines an operation on a class or object
  - Examples: getArea, speak, accelerate, eject

# Class vs. Program

- Last time...
  - We saw a class Wheel
- This time...
  - It has a “main” method
  - Program execution starts (and often ends) here
- Using the command line
  - Edit, compile, run (IntelliJ, javac, java)
  - See the byte code (javap -c)
- Examples: Robot.java and Calculator.java

# Wheel Class (simplified)

```
public class Wheel {
    double radius;

    Wheel(double radius) {
        this.radius = radius;
    }

    double getCircumference() {
        return 2 * Math.PI * radius;
    }

    double getArea() {
        return Math.PI * radius * radius;
    }

    double getRadius() {
        return radius;
    }
}
```

# Robot.java

```
public class Robot {  
    void speak(String message) {  
        System.out.println(message);  
    }  
  
    public static void main(String[] args) {  
        Robot r = new Robot();  
        r.speak("hello world");  
    }  
}
```

# Calculator.java

```
public class Calculator {  
    int add(int x, int y) {  
        return x + y;  
    }  
  
    int subtract(int x, int y) {  
        return x - y;  
    }  
  
    public static void main(String[] args) {  
        Calculator c = new Calculator();  
  
        System.out.println(c.add(3, 5));  
        System.out.println(c.subtract(3, 5));  
    }  
}
```

# Video 3

## Java Basics



# Java Basics

- Program structure
  - Top-level class
    - Name of class is name of file (with .java extension)
    - Example: class Robot in Robot.java
  - One or more methods
  - Program execution begins at main
- Command line input and output
  - Scanner class [not seen yet]
  - System.out.println(...)

# Syntax Details

- `//` or `/* */` for comments
- White space ignored (delimits tokens)
- Semicolon (`;`) to end or separate statements
- Curly braces (`{}`) to group statements
- Some words are “reserved” and cannot be used for variable, method, or class names
- `import` statement gives access to other classes

# System Class

- Has field members (variables) that reference the “standard input” and “standard output” files of the running Java program
- In command-line systems (e.g., UNIX shell), these “files” can be the keyboard and display, or redirected from disk files or other programs
- `System.in`: input stream
- `System.out`: output stream

# Parsing Input

- class Scanner
- To read from standard input, create a Scanner object and use its methods:

```
Scanner s = new Scanner(System.in);
```

```
String name = s.nextLine();
```

```
int i = s.nextInt();
```

```
double d = s.nextDouble();
```

# Calculator.java

```
import java.util.Scanner;

public class Calculator {
    int add(int x, int y) {
        return x + y;
    }

    int subtract(int x, int y) {
        return x - y;
    }

    public static void main(String[] args) {
        Calculator c = new Calculator();

        Scanner scanner = new Scanner(System.in);
        int x = scanner.nextInt();
        int y = scanner.nextInt();
        System.out.println(c.add(x, y));
    }
}
```

# Java Formatting Notes

- Naming conventions
  - Variables: lowerCamelCase
  - Classes: UpperCamelCase
  - Symbolic constants: UPPER\_CASE
- Open curly at end of line
- Use consistent indentation (e.g., 4 blanks)
- Complete documentation on course website

# Java Formatting Notes

honestaberoofing.com

What is this business?

HonestAbeRoofing.com

Oh ... this makes more sense!

# How to Read an Assignment (1)

When the assignment says, “Create a *class* Henway...”

- Create a file Henway.java that starts with this template...

```
public class Henway {  
}
```

- Note capitalization. Case matters!
- Compile—you should get no errors. If you run it, you will get an error (why?)



# How to Read an Assignment (2)

When the assignment says, “Write a Java program named Henway...”

- Create a file Henway.java that starts with this template...

```
public class Henway {  
    public static void main(String[] args) {  
        // Program execution begins here  
    }  
}
```

- Note capitalization. Case matters!
- Compile and run—you should get no errors (and no output!)

# How to Read an Assignment (3)

When the assignment says, “Read from standard input...”

- At the beginning of the file (before “public class ...”), insert...

```
import java.util.Scanner;
```

- At the beginning of the main method (for example), insert...

```
Scanner scanner = new Scanner(System.in);
```

- To read an integer and store it in variable x, use...

```
int x = scanner.nextInt();
```