Today

• A whirlwind tour of almost everything we have covered so far

• You should start preparing for mid-term if you haven’t
• Finish the mid-term practice before Wednesday
• Slides alone are not enough
• Review your lab / assignment code
Hardware vs. Software

• Hardware - physical machine
  – CPU, Memory
• Software - programs that give instructions to the computer
  – Windows XP, Games, Eclipse
Hardware

- CPU – the “brain” of your computer
- Memory – stores data for the computer
  - How much the “brain” can remember
  - Main memory - RAM
  - Auxiliary memory - Hard Drive

Levels of the Memory Hierarchy
Memory

• Measured in bytes
• 1 byte = 8 bits
• Bit is either 0 or 1
• Language of the computer is in bits
Programming Languages

- Your Program
- Compiler
- Machine Language (Bits)

High-level language (human readable)
Low-level language (computer readable)
Algorithms and pseudocode

• Algorithm – a set of instructions for solving a problem

• Pseudocode – combination of code and English used to express an algorithm before writing algorithm into code
Variables

• Used to store data in a program
• The data currently in a variable is its **value**
• Name of variable is an **identifier**
  – *Letters, digits, underscore*
  – Cannot start with digits
• Can change value throughout program
• Choose variable names that are meaningful!
How to use variables

• **Declare** a variable
  – `int number;`

• **Assign** a value to the variable
  – `number = 37;`

• **Change** the value of the variable
  – `number = 513;`
Keywords

• Reserved words with predefined meanings
• You cannot name your variables keywords
• if, else, return, new
Type

• What kind of value the variable can hold

• Two kinds of types.
  
  – **Primitive type** - indecomposable values
    • Names begin with lowercase letters
    • `int, double, char, float, byte, boolean, some others`

  – **Class type** - objects with both data and methods
    • Names by convention begin with uppercase letter
    • `Scanner, String, Student`
Assignment Statements

• Change a variable’s value
• Syntax:
  – variable = expression;
• Example:
  – sleepNeeded = 8;
  – sleepDesired = sleepNeeded * 2;
Assignment compatibilities

```
int x = 5;

double y = 12.7;
```

```
y = x;  \rightarrow  \Rightarrow  \text{ OK}
```

```
x = y;  \rightarrow  \Rightarrow  \text{ Not OK}
```
Type casting

\[ x = (\text{int}) y; \rightarrow \boxed{\text{OK}} \]
Arithmetic Operators

• Unary operators
  – +, -, ++, --, !

• Binary arithmetic operators
  – *, /, %, +, -
    • rate*rate + delta
    • 1/(time + 3*mass)
    • (a - 7)/(t + 9*v)
Modular Arithmetic - %

- **Remainder**
  - $7 \% 3 = 1$  \( (7 / 3 = 2, \text{ remainder } 1) \)
  - $8 \% 3 = 2$  \( (8 / 3 = 2, \text{ remainder } 2) \)
  - $9 \% 3 = 0$  \( (9 / 3 = 3, \text{ remainder } 0) \)
Parentheses and Precedence

• Expressions inside parentheses evaluated first
  – (cost + tax) * discount
  – cost + (tax * discount)
Errors

• *Syntax error* – grammatical mistake in your program
  – Java will not compile programs with syntax error

• *Run-time error* – an error that is detected during program execution
  – E.g., int avg = sum / total; // total is 0 in execution

• *Logic error* – a mistake in a program caused by the underlying algorithm
Strings

• A string (lowercase) is a sequence of characters
  – “Hello world!”
  – “Enter a whole number from 1 to 99.”

• String (capital S) is a class in Java, not a primitive type
String

String animal = “aardvark”;
System.out.println(animal);

aardvark
String Concatenation

String animal = “aardvark”;  
String sentence;  
sentence = “My favorite animal is the ” + animal;

My favorite animal is the aardvark
Strings methods

- `myString.length();`
- `myString.equals("a string");`
- `myString.toLowerCase();`
- `myString.trim();`
- Many others
String Indices

<table>
<thead>
<tr>
<th>U</th>
<th>N</th>
<th>C</th>
<th>i</th>
<th>s</th>
<th>G</th>
<th>r</th>
<th>e</th>
<th>a</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

`String output = myString.substring(1, 8);`
String Indices

<table>
<thead>
<tr>
<th>U</th>
<th>N</th>
<th>C</th>
<th>i</th>
<th>s</th>
<th>G</th>
<th>r</th>
<th>e</th>
<th>a</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

```
String output = myString.substring(1, 8);
```
## Escape Characters

<table>
<thead>
<tr>
<th>Escape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>Double quote</td>
</tr>
<tr>
<td>\’</td>
<td>Single quote</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
</tbody>
</table>
Scanner kb = new Scanner(System.in);
int num = kb.nextInt();
Comments

// this is a comment

/* This is also
   a comment.
   it ends
here --->*/
Boolean Expressions

• An expression that is either true or false
• Examples:
  – It is sunny today \((\text{true})\)
  – 10 is larger than 5 \((\text{true})\)
  – Today is Saturday \((\text{false})\)
if/else statements

Prompt user for integer

Is input greater than 10?

Yes

Print: “big number”

No

Print: “small number”

import java.util.*;

public class FlowChart
{
    public static void main(String[] args)
    {
        System.out.println("Give me an integer: ");
        Scanner keyboard = new Scanner(System.in);
        int inputInt = keyboard.nextInt();

        if (inputInt > 10)
        {
            System.out.println("big number");
        }
        else
        {
            System.out.println("small number");
        }
    }
}
If-else-if for multi-branch selection

if (case1) {
    // branch 1
}
else if (case2) {
    // branch 2
}
else if (case3) {
    ...
    ...
} else {
    ...
}

if (year==1) {
    System.out.println("Freshman");
} else if (year==2) {
    System.out.println("Sophomore");
} else if (year==3) {
    System.out.println("Junior");
} else {
    System.out.println("Senior");
}
## Java Comparison Operators for Primitive Values

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>Equal to</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Not equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

The result is a boolean value (true/false)

Example expressions:
- `variable <= 6`
- `myInt > 5`
- `5 == 3`
boolean type

- Can be either true or false

```java
boolean sunny = true;
boolean cloudy = false;

if (sunny || cloudy)
{
    // walk to school
}
```
&&, || operators

• AND

```java
if ((temperature > 50) && (temperature < 75))
{
    // walk to school
}
```

• OR

```java
if (sunny || cloudy)
{
    // walk to school
}
```
The ! (NOT) operator

- !true is false
- !false is true
- Example: walk to school if it is NOT cloudy

```java
if (!cloudy)
{
    // walk to school
}
```
Loops

• Loop: part of a program that repeats

• Body: statements being repeated

• Iteration: each repetition of body

• Stopping condition

Start

Enough sandwiches?

Yes

Distribute sandwiches

No

Make sandwich

- Sandwiches?
Types of Loops

• **while**
  – Safest choice
  – Not always most elegant

• **do-while**
  – Loop iterates AT LEAST once (not in exam.)

• **for**
  – Similar to **while**, but often more convenient syntax
  – Most useful when you have a known # of iterations you need to do
Using a **while** loop

```java
int n = 1;
while (n <= 10) {
    System.out.println(n);
    n = n + 1;
}
```

![Flowchart diagram](image)
Using a `for` loop

```java
int n;
for (n = 1; n <= 10; n++) {
    System.out.println(n);
}
```
Infinite loop example

```java
int n;
for (n = 1; n <= 10; n = 0)
{
    System.out.println(n);
}
```
The break statement

```java
for (int item = 1; item <= 5; item++)
{
    System.out.print("Enter cost of item "+
                     item + ": ");
    amount = keyboard.nextDouble();
    total = total + amount;
    if (total >= 100)
    {
        System.out.println("You spent all your money.");
        break;
    }
    System.out.println("Your total so far is "+total);
}
System.out.println("You spent "+total);
```
Ending a loop

• Count-controlled loops
  – If you know the number of loop iterations
  – `for (count = 0; count < iterations; count++)`

• User-controlled loops
  – Change the value of control variable
    • E.g., `Ask-before-iterating ( if user input is smaller than 0)`
    • E.g., Matching is found
Nested loops example: friendly greetings

```java
for (int i = 1; i < 10; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(i + "*" + j + "=" + (i * j) + "\t");
    }
    System.out.println();
}
```

- **Outer loop**: Iterates through `i` from 1 to 9.
- **Inner loop**: Iterates through `j` from 1 to `i`.

The code snippet prints a list of multiplication tables from 1 to 9, with each row starting from 1 and counting up to the current value of `i`.
Classes, Objects, and Methods

• Class: a definition of a kind of object
• Object: an instance of a class
  – Contains instance variables (data) and methods
• Methods
  – Methods that return a value
  – Methods that return nothing
Class

- A *class* is the definition of a kind of object
  - A blueprint for constructing specific objects

<table>
<thead>
<tr>
<th>Class Name:</th>
<th>Automobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data:</td>
<td>amount of fuel, speed, license plate</td>
</tr>
</tbody>
</table>
Objects, Instantiation

- **Object Name**: patsCar
  - Amount of fuel: 10 gallons
  - Speed: 55 miles per hour
  - License plate: “135 XJK”

- **Object Name**: suesCar
  - Amount of fuel: 14 gallons
  - Speed: 0 miles per hour
  - License plate: “SUES CAR”

- **Object Name**: ronsCar
  - Amount of fuel: 2 gallons
  - Speed: 75 miles per hour
  - License plate: “351 WLF”

Instantiations, or instances, of the class Automobile
Objects

• **Important**: classes do not have data; individual objects have data

• Classes specify what kind of data objects have
Creating an object

Create an object *anna* of class *Student*

```java
Student anna = new Student();
```

Create an object *keyboard* of class *Scanner*

```java
Scanner keyboard = new Scanner(System.in);
```
Instance variables

- Data defined in the class are called *instance variables*

```java
public String name;
public int classYear;
public double GPA;
public String major;
```

- *public*: no restrictions on how these instance variables are used (more details later – *public* is actually a bad idea here)

- *type*: int, double, String...
Methods

• Two kinds of methods
  – Methods that return a value
    • Examples: String’s .substring() method, String’s .indexOf() method, etc.
  – Methods that return nothing
    • Example: System.out.println()
Methods

```java
public String getMajor()
{
    return major;
}
```

- **public String getMajor()** returns a String
- **public void increaseYear()** returns nothing

```java
public void increaseYear()
{
    classYear++;  
}
```
Calling methods that return nothing

- object, followed by dot, then method name, then ()
- Use them as Java statements

```java
Student andrew = new Student();
andrew.classYear = 1;
andrew.increaseYear();
System.out.println("Andrew's class year is " + andrew.classYear);
```
Methods that return a value

```java
public String getClassYear()
{
    if (classYear == 1)
        return "Freshman";
    else if (classYear == 2)
        return "Sophomore";
    else if ...
}
```

Must return a value on every execution path
Calling methods that return a value

- object, followed by dot, then method name, then () (same as before)
- Use them as a value of the type specified by the method’s return type

```java
Student berkeley = new Student();
berkeley.major = “Studio Art”;

String m = berkeley.getMajor();
System.out.println(“Berkeley’s full name is ” + berkeley.getName());
System.out.println(“Berkeley’s major is ” + m);
```
Local/Instance variables

- **Instance variables**
  - Declared in a class
  - Confined to the class

- **Local variables**
  - Declared in a method
  - Confined to the method

```java
public class Student {
    public String name;
    public int classYear;
    public String major;

    public void printInfo(){
        String info = name + ":" + major + ":" + classYear;
        System.out.println(info);
    }

    public void increaseYear(int inc) {
        classYear += inc;
    }
}
```
public class Student {
    public String name;
    public int classYear;
    public String major;

    public void printInfo() {
        String info = name + "\t: " + major + "\t: " + classYear;
        System.out.println(info);
    }

    public void increaseYear(int inc) {
        classYear += inc;
    }
}
Simple example

public class Student {
    public String name;
    public int classYear;
    public String major;

    public void printInfo() {
        String info = name + ": ": " + major + ": ": " + classYear;
        System.out.println(info);
    }

    public void increaseYear(int inc) {
        classYear += inc;
        info = "info changed a bit";
    }
}
Methods with parameters

- Parameters are used to hold the value that you pass to the method

- Parameters can be used as (local) variables inside the method

```java
public int square(int number) {
    return number * number;
}
```

Parameters go inside parentheses of method header
Methods with multiple parameters

- Multiple parameters separated by commas

```java
public double getTotal(double price, double tax)
{
    return price + price * tax;
}
```
Methods with multiple parameters

• Multiple parameters separated by commas

```java
public class SalesComputer {
    public double getTotal(double price, double tax) {
        return price + price * tax;
    }
}
```

• **Order, type, and number** of arguments must match parameters specified in method header

```java
SalesComputer sc = new SalesComputer();
double total = sc.getTotal("19.99", Color.RED); // X
double total = sc.getTotal(19.99); // X
double total = sc.getTotal(19.99, 0.065);
int price = 50;
total = sc.getTotal(price, 0.065);
```
Calling methods from methods

• A method body can call another method
  – Done the same way:
    receiving_object.method();
• If calling a method of the same object, do not need
  receiving_object:
  – method();
• Alternatively, use the this keyword
  – this.method();
Several Common Mistakes

• Unwanted semicolon after if / for statements
  ```cpp
  if (a>b); // this semicolon causes an empty if-branch
  c++; // this line is always executed
  ```

  ```cpp
  for(int i = 0; i<10; i++); // this semicolon indicates an empty loop body
  c++; // this is executed only once
  ```

• Unpaired brackets
  Use indentation to help checking
  Use Eclipse’s auto format function
What is wrong with this code?

```c
int oddSum = 0;
int evenSum = 0;
for (int i = 1; i <= 6; i++)
{
    if (i % 2 == 0)
        evenSum = evenSum + i;
    else {
        oddSum = oddSum + i;
    }
}
```
What is wrong with this code?

```c
int oddSum = 0;
int evenSum = 0;
for (int i = 1; i <= 6; i++)
{
    if (i % 2 == 0)
    {
        evenSum = evenSum + i;
    }
    else
    {
        oddSum = oddSum + i;
    }
}
```
Indentation

• Indentation
  – Makes code easier to read
  – Helps with finding syntax and logic errors
  – Indent code that goes between { and }

• Be consistent!
Next Class

• Review Assignment 2
• Go through questions from mid-term practice worksheet
• Q&A

• Slides alone are not enough
• Review your lab / assignment code.