Routines

- Instructions
- Calls and returns
- Variables
- Passing parameters
- Expressions

```
main:
walkTo kitchen
CALL cleanFloors
CALL cleanWindows

walkTo livingRoom
CALL cleanFloors

walkTo bedroom
CALL cleanFloors
CALL cleanWindows
```

```
cleanFloors:
pick up trash
move furniture
CALL scrub
move furniture
RETURN
```

```
scrub:
spray cleaner
move cloth around
eat candy
RETURN
```

```
cleanWindows:
raise blinds
CALL scrub
lower blinds
RETURN
```

See Also

http://stackoverflow.com/

https://www.youtube.com/results?search_query=java+programming

https://docs.oracle.com/javase/tutorial/java/

Instructions

- Sequence of instructions
- End with a semicolon ";"
- Spacing does not matter
- Line and Block comments


```
System.out.println ("On that farm he had a cow.");

// This is a line comment
System.out.println("E I E I O"); // Ohio-i-o
System.out.println("On that farm he had a dog.");

/*
  This is a block
  comment
*/

      System.
    out
  .println(
"E I E I O");
```

Routines (Methods, Functions)

- Every routine has a name
  - Starts with lower case
  - Camel case
- Use braces to group instructions

```
singSong {

    System.out.println("On that farm he had a cow.");
    System.out.println("E I E I O");
    System.out.println("On that farm he had a pig.");
    System.out.println("E I E I O");

}
```

Routines (Methods, Functions)

- You can pass one or more parameters to a routine
  - Separate parameters with “,”
  - Give the type and name of each parameter
  - Use empty “( )” if there are none

```
singSong(String type, int count) {

    System.out.println("On that farm he had a cow.");
    System.out.println("E I E I O");
    System.out.println("On that farm he had a pig.");
    System.out.println("E I E I O");

}
```

Routines (Methods, Functions)

- A function can return ONE value
  - Only define the type
  - Use “void” to indicate returning nothing

```
void singSong(String type, int count) {

    System.out.println("On that farm he had a cow.");
    System.out.println("E I E I O");
    System.out.println("On that farm he had a pig.");
    System.out.println("E I E I O");

}
```

Routines (Methods, Functions)

- You will use other attributes before the return type
- We will discuss these shortly

```
public static void singSong(String type, int count) {

    System.out.println("On that farm he had a cow.");
    System.out.println("E I E I O");
    System.out.println("On that farm he had a pig.");
    System.out.println("E I E I O");

}
```

Calling Routines

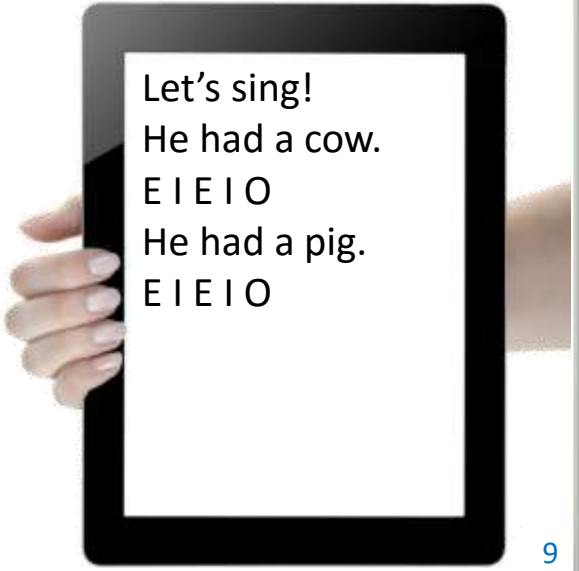
- You "call" a routine by using its name as an instruction
- The computer stops where it is and jumps to the routine
- When the routine is done it "returns" to where it was before

```
4⊖    public static void singEIEIO() {
5         System.out.println("E I E I O");
6     }
7
8⊖    public static void singSong() {
9         System.out.println("He had a cow.");
10        singEIEIO();
11        System.out.println("He had a pig.");
12        singEIEIO();
13    }
14
```

```java
public class Farm {

    public static void singEIEIO() {
        System.out.println("E I E I O");
    }

    public static void singSong() {
        System.out.println("He had a cow.");
        singEIEIO();
        System.out.println("He had a pig.");
        singEIEIO();
    }

    public static void main(String[] args) {
        System.out.println("Let's sing!");
        singSong();
    }

}
```

Class

- Collection of routines
- Class names start upper case
- Order does not matter
- The "main" is the first routine



Class Variables

```java
public class Tinker {

  public static int sum = 0;

  public static void doStuff() {
    sum = sum + 1;
  }

  public static void doOther() {
    sum = sum * 10;
  }

  public static void main(String [] args) {
    sum = 2;      // sum is now 2
    doStuff();    // sum is now 3
    doOther();    // sum is now 30
    System.out.println(sum);
  }
}
```

- Declared along with routines
- Type, name, and initial value
- Shared by all routines

Local Variables

```java
public class Tinker2 {

  public static void doStuff() {
    int a = 20;
    a = a * 5;
    System.out.println(a); // Prints "100"
  }

  public static void main(String [] args) {
    int a = 5;
    System.out.println(a); // Prints "5"
    doStuff();
    System.out.println(a); // Prints "5"
  }

}
```

- Declared inside a routine
- Only available to that routine
- Created when the routine is called
- Destroyed when the routine returns
- MUST be initialized before use
- Can be declared anywhere in the routine

Passing Parameters

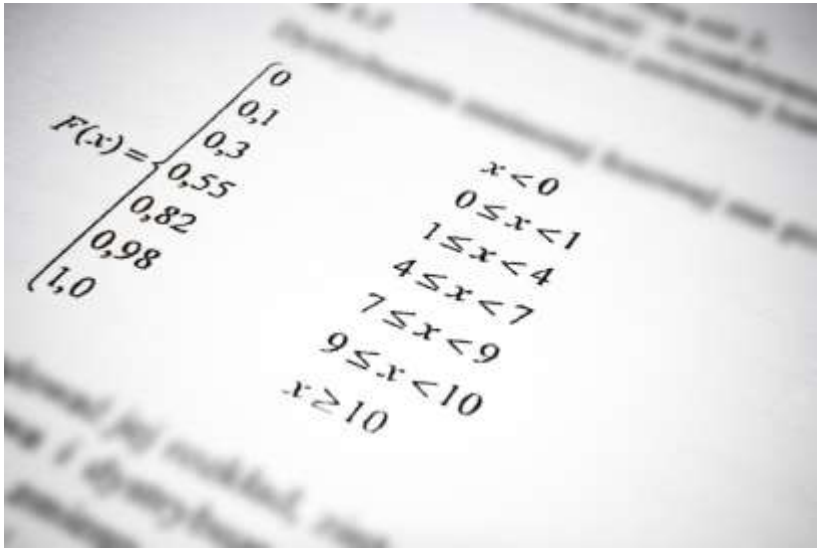
```java
public class Tinker3 {

  public static void doStuff(int a) {
    // int a; The parameters are locals
    a = a * 10;
    System.out.println(a);
  }

  public static void main(String [] args) {
    int a = 5;
    doStuff(20);
    doStuff(a);
    System.out.println(a);
  }

}
```

- You can pass parameters to routines
- The values are COPIED into locals
- The routine CANNOT change the caller's variable

```java
public class Tinker4 {

  public static int addThem(int a, int b) {
    int c = a + b;
    return c;
  }

  public static void main(String [] args) {
    int a = addThem(3,7);
    System.out.println(a);
    a = addThem(a,2);
    System.out.println(a);
    addThem(100,100);
  }

}
```

Return Values

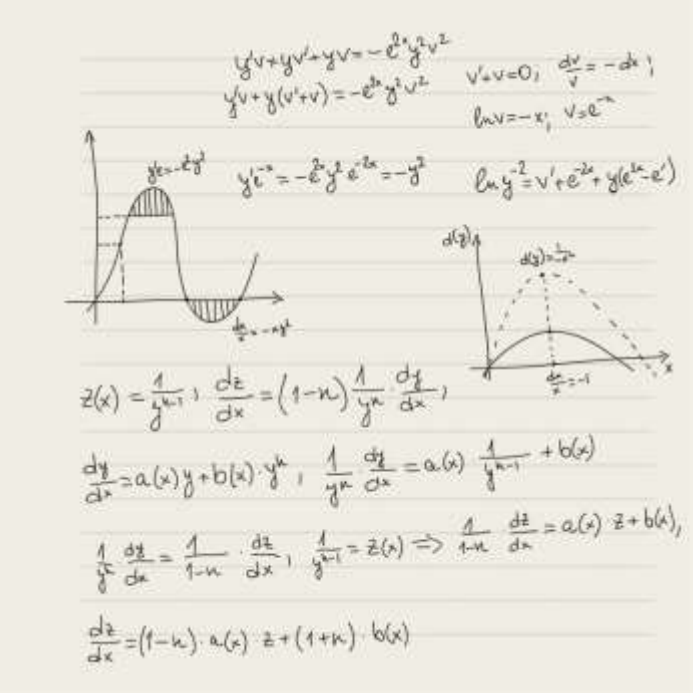
- Routines can return one value
- The return is copied to the caller
- You can ignore the return value



Expressions

- Variables and constants can be used interchangeably
- Return values can be used as terms
- Consider breaking things up into steps

```
int a = addThem(1,2); // 3
int b = addThem(3,4); // 7
int x = addThem(a,b); // 10

int a = addThem(1,2); // 3
int x = addThem( a , addThem(3,4) ); // 10

int x = addThem( addThem(1,2) , addThem(3,4) );

int x = addThem(a, addThem(3,4)) * 2 +
                addThem(a,a);
```



Overloading

- Methods can have the same name
- Must have different arguments
- Return type does not count

```
public static void doStuff(int a, int b) {
            System.out.println("TWO");
}

public static void doStuff(int a) {
            System.out.println("INTEGER");
}

public static void doStuff(short b) {
            System.out.println("SHORT");
}

doStuff(1,2);          // TWO
doStuff(10);           // INTEGER
doStuff((short)10);    // SHORT
```

Tinkering

- Type in the Farm class and run it.
- Try making the song more complete. What helper routines would make the logic easier to code?
- Type in the Tinker4 class and run it.
- Try making a routine that calls itself. Can a routine call itself over and over forever? What runtime error do you get? What's going on?