Lecture 7: Methods

AP Computer Science Principles

Declaring a method

- A **method** is a sequence of related statements performing a specific task.
- **Declaring a method** is giving a method a name so that it can be executed. We have seen two built-in methods draw() and setup().
-

```
void name() {
    statement;
    statement;
    ...
    statement;
}
```

- Example:

```
void printWarning() {
   println("This product causes cancer in humans.");
}
```

Calling a method

Executes the method's code

- Syntax:

```
name();
```

  - You can call the same method many times if you like.

- Example:

```
printWarning();
printWarning();
```

  - Output:

```
This product causes cancer in humans.
This product causes cancer in humans.
```

Control flow

- When a method is called, the program's execution...
  - "jumps" into that method, executing its statements, then
  - "jumps" back to the point where the method was called.

Methods calling methods

```
void setup() {
        message1();
        message2();
        println("Done with setup.");
}

void message1() {
        println("This is message1.");
}

void message2() {
        println("This is message2.");
        message1();
        println("Done with message2.");
}
```

Control flow

- When a method is called, the program's execution...
  - "jumps" into that method, executing its statements, then
  - "jumps" back to the point where the method was called.

```
void setup() {
        message1();

        message2();

        println("Done wit

    }


    ...
}
```

```
void message1() {
    println("This is message1.");
}
```

```
void message2() {
    println("This is message2.");
    message1();
    ntln("Done with message2.");
}
```

```
    essage1() {
    println("This is message1.");
}
```

Methods calling methods

```
void setup() {
        message1();
        message2();
        println("Done with setup.");
}
void message1() {
        println("This is message1.");
}
void message2() {
        println("This is message2.");
        message1();
        println("Done with message2.");
}
```

- Output:

```
This is message1.
This is message2.
This is message1.
Done with message2.
Done with setup.
```

When to use methods

- Place statements into method if:
  - The statements are related structurally, and/or
  - The statements are repeated.

- You should not create methods for:
  - Unrelated or weakly related statements.
    (Consider splitting them into two smaller methods.)

Modularity

Modularity

- **modularity**: Writing code in smaller, more manageable components or modules. Then combining the modules into a cohesive system.
  - Modularity with methods. Break complex code into smaller tasks and organize it using methods.

Modularity with Methods

```
int centerX, centerY, xspeed, yspeed;
void setup(){
        size(800,600);
        centerX = 400; centerY = 300;
        xspeed = 5; yspeed = 4;
}
void draw(){
        background(255);
        fill(255,0,0);
        ellipse(centerX,centerY,80,80);
        centerX = centerX + xspeed;
        centerY = centerY + yspeed;
        if (centerX > width || centerX < 0)
                xspeed = -xspeed;
        if (centerY > height || centerX < 0)
                yspeed = -yspeed;
}
```

How do we make this modular?

Break into smaller tasks and organize code using functions.

Modularity with Methods

```
int centerX, centerY, xspeed, yspeed;
void setup(){
        size(800,600);
        centerX = 400; centerY = 300;
        xspeed = 5; yspeed = 4;
}
void draw(){
        background(255);
        fill(255,0,0);
        ellipse(centerX,centerY,80,80);
        centerX = centerX + xspeed;
        centerY = centerY + yspeed;
        if (centerX > width || centerX < 0)
                xspeed = -xspeed;
        if (centerY > height || centerX < 0)
                yspeed = -yspeed;
}
```

How do we make this modular?

Break into smaller tasks and organize code using functions.

Modularity with Methods

```
int centerX, centerY, xspeed, yspeed;
void setup(){
        size(800,600);
        centerX = 400; centerY = 300;
        xspeed = 5; yspeed = 4;
}
void draw(){
        background(255);
        drawBall();
        centerX = centerX + xspeed;
        centerY = centerY + yspeed;
        if (centerX > width || centerX < 0)
                xspeed = -xspeed;
        if (centerY > height || centerX < 0)
                yspeed = -yspeed;
}
```

How do we make this modular?

Break into smaller tasks and organize code using functions.

Modularity with Methods

```
int centerX, centerY, xspeed, yspeed;
void setup(){
        size(800,600);
        centerX = 400; centerY = 300;
        xspeed = 5; yspeed = 4;
}
void draw(){
        background(255);
        drawBall();
        centerX = centerX + xspeed;
        centerY = centerY + yspeed;
        if (centerX > width || centerX < 0)
                xspeed = -xspeed;
        if (centerY > height || centerX < 0)
                yspeed = -yspeed;
}
```

How do we make this modular?

Break into smaller tasks and organize code using functions.

Modularity with Methods

```
int centerX, centerY, xspeed, yspeed;
void setup(){
        size(800,600);
        centerX = 400; centerY = 300;
        xspeed = 5; yspeed = 4;
}
void draw(){
        background(255);
        drawBall();
        moveBall();
        if (centerX > width || centerX < 0)
                xspeed = -xspeed;
        if (centerY > height || centerX < 0)
                yspeed = -yspeed;
}

}
```

How do we make this modular?

Break into smaller tasks and organize code using functions.

Modularity with Methods

```
int centerX, centerY, xspeed, yspeed;
void setup(){
        size(800,600);
        centerX = 400; centerY = 300;
        xspeed = 5; yspeed = 4;
}
void draw(){
        background(255);
        drawBall();
        moveBall();
        if (centerX > width || centerX < 0)
                xspeed = -xspeed;
        if (centerY > height || centerX < 0)
                yspeed = -yspeed;
}

}
```

How do we make this modular?

Break into smaller tasks and organize code using functions.

Modularity with Methods

```
int centerX, centerY, xspeed, yspeed;
void setup(){
        size(800,600);
        centerX = 400; centerY = 300;
        xspeed = 5; yspeed = 4;
}
void draw(){
        background(255);
        drawBall();
        moveBall();
        bounceBall();
}
```

How do we make this modular?

Break into smaller tasks and organize code using functions.

Modularity with Methods

```
int centerX, centerY, xspeed,
yspeed;
void setup(){…}
void draw(){
  background(255);
  drawBall();
  moveBall();
  bounceBall();
}
void drawBall(){
 fill(255,0,0);
 ellipse(centerX,centerY,80,80);
}
```

```
void moveBall(){
  centerX = centerX + xspeed;
  centerY = centerY + yspeed;
}

void bounceBall(){
  if(centerX>width||centerX<0)
     xspeed = -xspeed;
  if(centerY>height||centerX<0)
     yspeed = -yspeed;
}
```

Modularity helps with debugging. For example, if the ball is not bouncing properly, we only need to look at the bounceBall method.

Method Parameters

Methods in Java can have parameters or input arguments.

Syntax:

```
void methodName(type var1)
{…} // declare
```

The parameters in the method header are **formal parameters.**

```
methodName(expression); // call
```

The parameters in the method header are **actual parameters.**

Method Parameters

Example:

```
void printDouble(int x)
{
      println(“Your number ” + x
                  + “doubled is ” + 2 * x + ”.”);
}
```

To call:

```
printDouble(5); //Your number 5 doubled is 10.
printDouble(3.4); //Error! Incompatible types!
```

Method Parameters

```
void setup(){
    printDouble(12); //Your number 12 doubled is 24.
}

void printDouble(int x)
{
      println(“Your number ” + x
                  + “doubled is ” + 2 * x +”.”);
}
```

Return

- **return**: To send out a value as the result of a method.
  - The opposite of a parameter:
    - Parameters send information *in* from the caller to the method.
    - Return values send information *out* from a method to its caller.
      - A call to the method can be used as part of an expression.

Method Parameters

Methods in Java can have return types.

```
type methodName(type var1,…, type var2)
{ <statements>} // declare
```

Example:

```
void draw(){
        int a = doubleThis(9); //a is now 18

}
int doubleThis(int n)
{
        int m=2*n;
        return m;
}
```

Method Parameters

```
void setup(){
	int x = 5;
	int a = doubleThis(x);
	println("My number doubled is " + a);
	//My number doubled is 10
}

int doubleThis(int n)
{
	int m=2*n;
	return m;
	//or simply, return 2*n;
}
```

Return

```
void setup(){
      int x=6, y;
      doubleThis(x); //returned value NOT SAVED!
      y = doubleThis(x); // y is now 12.
      println("x doubled is " + y);
      println("x is now " + x);
      println("y doubled is " + doubleThis(y));
}
```

Output:

```
x doubled is 12
x is now 6
y doubled is 24
```

Value semantics

- **value semantics**: When primitive variables (`int`, `float`, `boolean`) are passed as parameters, their values are copied.
  - Modifying the parameter will not affect the variable passed in.

```
void setup() {
    int x = 23;
    strange(x);
    println("2. x = " + x);
}
void strange(int x) {
   x = x + 1;
   println("1. x = " + x);
}
```

Output:

```
1. x = 24
2. x = 23
```

Value semantics

Value semantics: methods cannot change the values of primitive types(`int, boolean, float`) and `String`.

```
void setup(){
	int x = 5;
	doubleMyNumber(x);
	println("My number is" + x); //My number is 5
}
void doubleMyNumber(int x)
{
	x = 2 * x;
}
```

Note: The value of x did not change. The x variable in setup() is **NOT** the same x variable in doubleMyNumber.

Multiple Inputs

```
void setup() {
    int x = 3, y = 10;
    strange(x,y);
    strange(y,x);
}
void strange(int x, int y) {
    println(x + " " + y);
}
```

Output:

```
3 10
10 3
```

Common error: Not storing

- Many students incorrectly think that a `return` statement sends a variable's name back to the calling method.

```
void setup{
    slope(1, 0, 6, 3);
    println("The slope is " + result);  // ERROR:
}                                       // result not defined

float slope(int x1, int x2, int y1, int y2) {
    float dy = y2 - y1;
    float dx = x2 - x1;
    float result = dy / dx;
    return result;
}
```

Fixing the common error

- Instead, returning sends the variable's *value* back.
  - The returned value must be stored into a variable or used in an expression to be useful to the caller.

```
void setup() {
    float s = slope(1, 0, 6, 3);
    println("The slope is " + s);
  // OR println("The slope is " + slope(1,0,6,3));
}
float slope(int x1, int x2, int y1, int y2) {
    float dy = y2 - y1;
    float dx = x2 - x1;
    float result = dy / dx;
    return result;
}
```

Return examples

```
// Computes triangle hypotenuse length given its side
   lengths
float hypotenuse(int a, int b) {
    float c = sqrt(a * a + b * b);
    return c;
}
```

- You can shorten the examples by returning an expression:

```
float hypotenuse(int a, int b) {
    return sqrt(a * a + b * b);
}
```

Return String type

```
void draw(){
   println(4 + " is " + evenOdd(4)); // 4 is even
   println(35 + " is " + evenOdd(35); // 35 is odd


}
String evenOdd(int a) {
   if(a % 2 == 0){
       return "even";
   else{
       return "odd";
   }
}
```

Method Calls

- A method that is declared but not called will not be executed!

```
void setup{
   method1();
}
void draw(){
   method2();
   method1();
}

void method1() {
   println("this is method1");
}
int method2(){
    println("this is method2");
}
void method3(){
    println("this is method3");}
```

`method3()` is never called and will never be executed.

Errors Example

```
void draw(){
      twice(4);// 8 but returned value not stored! Avoid!
      int a = twice(5);// a=10, returned value saved in a
      twice(); // error! missing input
      println(twice(2)); // 4 is printed
      twice(2,3); // error! too many inputs
      twice(3.5); // error! wrong input type
}
int twice(int a){
      return 2 * a;
}
```

Lab 1

Modify the bouncing ball program to use methods as shown in this lecture.

```
int centerX, centerY, xspeed, yspeed;
void setup(){
        size(800,600);
        centerX = 400; centerY = 300;
        xspeed = 5; yspeed = 4;
}
void draw(){
        background(255);
        drawBall();
        moveBall();
        bounceBall();
}
```

Lab 1 Outline

```
void setup(){…}

void draw(){…}

void drawBall(){… }

void moveBall(){… }

void bounceBall(){… }
```

Lab 2: Day Of the Week

Write a program that outputs the day of the week for a given date.

Given the month m, day d and year y, the day of the week(Sunday = 0, Monday = 1, …, Saturday = 6) D is given by:

$$
\begin{aligned}
y_0 &= y - (14 - m)/12 \\
x_0 &= y_0 + y_0/4 - y_0/100 + y_0/400 \\
m_0 &= m + 12 \times ((14 - m)/12) - 2 \\
\mathcal{D} &= (d + x_0 + 31 \times m_0/12) \bmod 7
\end{aligned}
$$

}

Lab 2: Day Of the Week

Write the draw() method so that the output is similar to the following:

Date: 9-25-2018

Day of the week: Tuesday

Your program must contains the following two methods:

```
// day returns D from the formula.
int day(int m, int d, int y){…}
// dayOfWeek returns “Monday”, “Tuesday” etc..
String dayOfWeek(int D){…}
```

Lab 2 Outline

```
void setup(){…}

void draw(){…}

int Day(int m, int d, int y){…}

// this method must call the Day method
// returns “Sunday” if D = 0, “Monday” if D = 1, etc…
String dayOfWeek(int D){…}
```

Lab 3: BMI

Formula for body mass index (BMI):

$$BMI = \frac{weight}{height^2} \times 703$$

BMI	Weight class
below 18.5	underweight
[18.5 – 25)	normal
[25.0 – 30)	overweight
30.0 and up	obese

- Write a program that produces output like the following:

```
Height (in inches): 70.0
Weight (in pounds): 194.25
BMI = 27.868928571428572
Overweight
```

Lab 3: BMI

Your program must include two methods:1) the method `bmi` which takes two float parameters height and weight and returns the bmi and 2) the method `weightClass` which takes two float parameters height and weight and returns a string classifying the weight class. **The weightClass method must call the bmi method!**

```
float bmi(float height, float weight)
{…}
String weightClass(float height, float weight)
{…}
```

Lab 3 Outline

```
void setup(){…}

void draw(){…}

float bmi(float height, float weight)
{…}

String weightClass(float height, float weight)
{…}
```

Homework

1) Read and reread these lecture notes.

2) Complete the problem set.

3) Complete the labs.

References

Part of this lecture is taken from the following book.

Stuart Reges and Marty Stepp. Building Java Programs: A Back to Basics Approach. Pearson Education. 2008.