Methods

Creating Computational Abstractions

How methods work

- A method is a block of program code that can be called from other parts of your program.
- Methods may require arguments when they are executed to provide data for the method to use.
- Methods may return a computed result back to the program statement that calls the method.
- Example method signatures from the `String` API:
  - **`int length()`**
  - **`char charAt(int position)`**
  - **`String substring(int beginIndex, int endIndex)`**

About methods

- Every Java program starts execution with the `main` method.
- The main method can call other methods to do part of the work of the program.
  - These methods can be part of other classes.

  and
  - These methods can be part of the class containing `main`.
- Static vs. non-static methods
  - Methods called on objects are non-static.
  - Methods called directly without creating objects are static.

Writing static methods

`public static` *return-type method-name* `(` *parameter-list* `)`

parameter-list

- A list of variables (and their data types) to hold data that is passed to the method when it is called.
- This list can be empty.

return-type

- A data type indicating the type of the data that is returned back to the instruction that calls this method.
- If no data is returned back, use **`void`**.

Example

```
public static void displayQuestion()
{
  System.out.println
     ("What food does Homer like?");
}
```


Another Example

```
public static void displayScores
  (int score1, int score2)
{
  System.out.println("***************");
  System.out.println("Player 1: "+ score1);
  System.out.println("Player 2 :"+ score2);
  System.out.println("***************");
}
```

Calling These Methods

```
public static void main(String[] args) {
  // examples
  displayQuestion();
  System.out.println("DONUTS!");
  displayScores(10, 3);
  int value = 12;
  displayScores(value, value+4);

}
```


`return` statement

`return` *value* `;`

- Returns the indicated value to the instruction that called this method.
- The value can be a variable or an expression.
- The data type of the indicated value must match the return type indicated in the signature of the method.
- If a `return` statement is executed, control passes back to the calling method immediately. (Any remaining instructions in this method are not executed.)

Example

parameter

```
public static int computeSum(int n) {
  // computes and returns the sum of
  // 1 + 2 + ... + n assuming n > 0
  int sum = n * (n+1) / 2;
  return sum;
}
```

a variable declared in a method is called a local variable (its scope is only the method where it is declared)

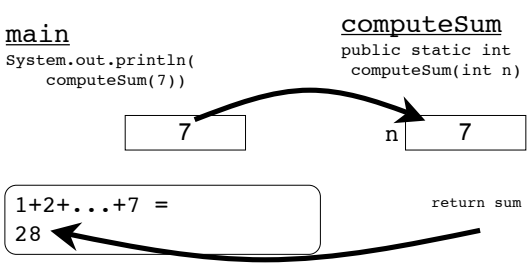
Calling computeSum

```
public static void main(String[] args) {
  System.out.println("1+2+3+4+5+6+7 = ");
  System.out.println(computeSum(7));
}
```

`static` method `computeSum` goes here, after `main`

argument

How it works



Another example

```
public static double computeAvgGrade
  (int total, int numGrades)
{
  if (numGrades <= 0)
     return -1.0;
  return (double)total/numGrades;
}
```

parameters

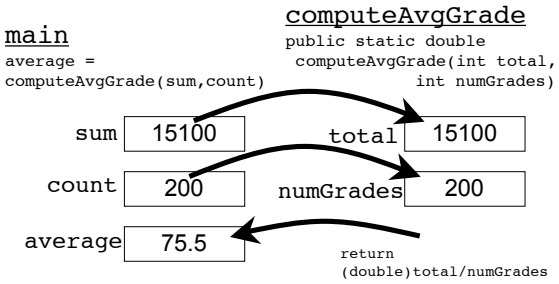
Calling computeAvgGrade

(from another method in the same class, like `main`)

```
System.out.println(
  "Input sum of all scores: ");
int sum = scan.nextInt();
System.out.println(
  "Input number of scores: ");
int count = scan.nextInt();
double average =
        computeAvgGrade(sum, count);
System.out.println(average);
```

arguments

How it works



Exercise 1

```
public static int findMax(int x, int y) {
  // returns the maximum of x and y




}
```


Exercise 2

Write a code fragment that asks the user for two integers and outputs the maximum of the two integers using the method you just wrote in the previous exercise.

```
Scanner scan = new Scanner(System.in);
System.out.println("Input first number");
int num1 = scan.nextInt();
System.out.println("Input second number");
int num2 = scan.nextInt();
System.out.println("The maximum is " +

                   _____________________________________);
```


Calling non-void methods

- If you call a method that has a non-void return type, you should indicate in your instruction what you will do with the returned data.
  - Print it out.

    ```
    System.out.println(computeSum(n));
    ```
  - Store the result in a variable for later use.

    ```
    average =
      computeAvgGrade(total, numGrades);
    ```


Calling void methods

- If you call a method that has a `void` return type, you must call this method by itself as an instruction - not embedded in another operation.
  - Correct:

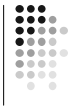
    ```
    displayQuestion();
    displayScores(15, 100);
    ```
  - Incorrect:

    ```
    System.out.print(displayQuestion());
    total = displayScores(15, 100);
    ```

Overloading

- Methods can have the same name but different parameter lists (number of parameters, types of parameters)
- The compiler can figure out which method you are calling based on how many and the types of the arguments that you supply when you call the overloaded method.
- Example: `substring` in `String`

Overloading Example

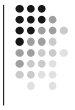


```
public static int computeSum(int x, int y)
{
  // returns x+(x+1)+...+y assuming y > x
  // and x > 0 and y > 0
  int sum1 = y * (y+1) / 2;  // 1+...+y
  int sum2 = (x-1) * x / 2;  // 1+...+(x-1)
  return sum1 - sum2 ;
}
```


Another Way

```
public static int computeSum(int x, int y)
{
  // returns x+(x+1)+...+y assuming y > x
  // and x > 0 and y > 0
  int sum1 = computeSum(y);  // 1+...+y
  int sum2 = computeSum(x-1);// 1+...+(x-1)
  return sum1 - sum2 ;
}
```


Calling computeSum



```
public static void main(String[] args)
{
  System.out.println("1+2+3+4+5+6+7 = ");
  System.out.println(computeSum(7));
  System.out.println("5+6+7+8+9+10 = ");
  System.out.println(computeSum(5,10));
}
```


Overloading ambiguity

```
public static int compute(int x, int y)
{
  ....
}
public static int compute(int a, int b)
{
  ....
}
```

compiler sees these as the same parameter list (2 `int`s) - SYNTAX ERROR