Inf1-OP
Functions aka Static Methods

Volker Seeker, adapting earlier version by Perdita Stevens and Ewan Klein

School of Informatics

February 26, 2018
Functions / Static Methods
Why are functions so helpful?
Find Nearest Neighbour to a Central Point

Sequence of x-y point coordinates as arguments to program
class NearestNeighbourBad {
    public static void main(String[] args) {
        int N = args.length;
        if (N % 2 != 0) N--; // ignore final arg if odd number
        double[] points = new double[N];

        for (int i = 0; i < N; i++)
            points[i] = Double.parseDouble(args[i]);

        double[] centre = { points[0], points[1] }; // first point is our centre
        System.out.printf("Centre lies at (%5.2f, %5.2f)\n", centre[0], centre[1]);

        double[] neighbours = new double[points.length - 2];
        for (int i = 2; i < points.length; i++) // all except the first are neighbours
            neighbours[i - 2] = points[i];

        double[] dists = new double[neighbours.length / 2];
        for (int i = 0; i < neighbours.length; i += 2) { // step over two at a time to get x and y
            double d1 = centre[0] - neighbours[i];
            double d2 = centre[1] - neighbours[i + 1];
            dists[i / 2] = Math.sqrt(d1*d1 + d2*d2);
        }

        for (int i = 0; i < dists.length; i++)
            System.out.printf("Distance to (%5.2f, %5.2f) is %5.2f\n",
                neighbours[(i*2)], neighbours[(i*2) + 1], dists[i]);

        double min = dists[0];
        for (int i = 1; i < dists.length; i++)
            if (dists[i] < min) min = dists[i];

        System.out.printf("Minimum distance to centre is %5.2f\n", min);
    }
}
This is Terrible!
This is Terrible!

But don’t worry. Functions will make this so much easier!
What do we need to do?

- Parse arguments
- Get centre
- Print centre
- Get neighbours
- Calculate distances
- Print distances
- Calculate minimum
- Print minimum

Let's think about what we need for those steps.
What do we need to do?

- parse arguments
What do we need to do?

▶ parse arguments
▶ get centre
What do we need to do?

- parse arguments
- get centre
- print centre
What do we need to do?

- parse arguments
- get centre
- print centre
- get neighbours
What do we need to do?

▶ parse arguments
▶ get centre
▶ print centre
▶ get neighbours
▶ calculate distances
What do we need to do?

- parse arguments
- get centre
- print centre
- get neighbours
- calculate distances
- print distances
What do we need to do?

- parse arguments
- get centre
- print centre
- get neighbours
- calculate distances
- print distances
- calculate minimum
What do we need to do?

- parse arguments
- get centre
- print centre
- get neighbours
- calculate distances
- print distances
- calculate minimum
- print minimum
What do we need to do?

- parse arguments
- get centre
- print centre
- get neighbours
- calculate distances
- print distances
- calculate minimum
- print minimum

Let's think about what we need for those steps.
What do we need to do?

- parse arguments
- get centre
- print centre
- get neighbours
- calculate distances
- print distances
- calculate minimum
- print minimum

Let's think about what we need for those steps.
What do we need to do?

- points ← parse arguments ← arguments
- get centre
- print centre
- get neighbours
- calculate distances
- print distances
- calculate minimum
- print minimum

Let's think about what we need for those steps.
What do we need to do?

- points ← parse arguments ← arguments
- centre ← get centre ← points
- print centre
- get neighbours
- calculate distances
- print distances
- calculate minimum
- print minimum

Let's think about what we need for those steps.
What do we need to do?

- points ← parse arguments ← arguments
- centre ← get centre ← points
- print centre ← arguments
- get neighbours
- calculate distances
- print distances
- calculate minimum
- print minimum

Let's think about what we need for those steps.
What do we need to do?

- points ← parse arguments ← arguments
- centre ← get centre ← points
- print centre ← arguments
- neighbours ← get neighbours ← points
- calculate distances
- print distances
- calculate minimum
- print minimum

Let's think about what we need for those steps.
What do we need to do?

- points ← parse arguments ← arguments
- centre ← get centre ← points
- print centre ← arguments
- neighbours ← get neighbours ← points
- distances ← calculate distances ← centre, neighbours
- print distances
- calculate minimum
- print minimum

Let's think about what we need for those steps.
What do we need to do?

- points ← parse arguments ← arguments
- centre ← get centre ← points
- print centre ← arguments
- neighbours ← get neighbours ← points
- distances ← calculate distances ← centre, neighbours
- print distances ← distances
- calculate minimum
- print minimum

Let's think about what we need for those steps.
What do we need to do?

- points ← parse arguments ← arguments
- centre ← get centre ← points
- print centre ← arguments
- neighbours ← get neighbours ← points
- distances ← calculate distances ← centre, neighbours
- print distances ← distances
- minimum ← calculate minimum ← distances
- print minimum

Let's think about what we need for those steps.
What do we need to do?

- points ← parse arguments ← arguments
- centre ← get centre ← points
- print centre ← arguments
- neighbours ← get neighbours ← points
- distances ← calculate distances ← centre, neighbours
- print distances ← distances
- minimum ← calculate minimum ← distances
- print minimum ← minimum

Let's think about what we need for those steps.
WHAT IF I TOLD YOU

THIS IS ALL YOU NEED
public static void main(String[] args) {
    double[] points = parseArguments(args);
    double[] centre = getCentre(points);
    printCentre(centre);
    double[] neighbours = getNeighbours(points);
    double[] distances = calcDistances(centre, neighbours);
    printDistances(distances, neighbours);
    double minimum = calcMinimum(distances);
    printMinimum(minimum);
}
Main Function for Nearest Neighbour

```java
public static void main(String[] args) {
    double[] points = parseArguments(args);
    double[] centre = getCentre(points);
    printCentre(centre);
    double[] neighbours = getNeighbours(points);
    double[] distances = calcDistances(centre, neighbours);
    printDistances(distances, neighbours);
    double minimum = calcMinimum(distances);
    printMinimum(minimum);
}
```

This is simply what we just developed plus some types and brackets.
All that is left to do is write some simple functions.
All that is left to do is write some simple functions.

So what does a function look like?
Euclidian Distance between two Points

- Given some ‘special’ point \( p \), how close are various other points to \( p \)?
- Useful, for example, if trying to find the closest point to \( p \).
- Use Euclidean distance — restricted to 2D case, where \( p = (p_0, p_1) \) etc.:

\[
\text{dist}(p, q) = \sqrt{(p_0 - q_0)^2 + (p_1 - q_1)^2}
\]
Euclidian Distance between two Points

- Given some ‘special’ point $p$, how close are various other points to $p$?
- Useful, for example, if trying to find the closest point to $p$.
- Use Euclidean distance — restricted to 2D case, where $p = (p_0, p_1)$ etc.

$$dist(p, q) = \sqrt{(p_0 - q_0)^2 + (p_1 - q_1)^2}$$

```java
public static double distance(double x0, double y0,
                                 double x1, double y1) {
    double d1 = x0 - x1;
    double d2 = y0 - y1;
    return Math.sqrt(d1*d1 + d2*d2);
}
```
public static double distance (double x0, double y0, 
double x1, double y1) {
    double d1 = (x0 - x1);
    double d2 = (y0 - y1);
    return Math.sqrt(d1*d1 + d2*d2);
}
Anatomy of a Java Function

```java
public static double distance(double x0, double y0, double x1, double y1) {
    double d1 = (x0 - x1);
    double d2 = (y0 - y1);
    return Math.sqrt(d1*d1 + d2*d2);
}
```
Calling a Function

Literal arguments

double d = distance(3.0, 5.0, 14.25, 2.70);

Variable arguments

double p0 = 3.0;
double p1 = 5.0;
double q0 = 14.25;
double q1 = 2.70;

double d = distance(p0, p1, q0, q1);
Flow of Control with Functions

Schematic Structure of Program

```java
public class PointDistance {

    public static double distance(double x0, double y0,
                               double x1, double y1) {
        ...
    }

    public static void main(String[] args) {
        ...
        double dist = distance(p0, p1, q0, q1);
        ...
    }
}
```
Flow of Control with Functions

Functions provide a **new way** to control the flow of execution.

What happens when a function is called:

- Control transfers to the code in body of the function.
- Parameter variables are assigned the values given in the call.
- Function code is executed.
- Return value is assigned in place of the function call in the calling code.
- Control transfers back to the calling code.
Pass by Value

- **Pass by Value**: parameter variables are assigned the values given by arguments to the call.
- The function only has access to the values of its arguments, not the arguments themselves.
- Consequently, changing the value of an argument in the body of the code has no effect on the calling code.
Pass by Value

```java
public class AddOne {
    public static void addOne(int num) {
        num++;
    }
    public static void main(String[] args) {
        int x = 0;
        addOne(x);
        System.out.println(x);
    }
}
```

Output

```bash
% java AddOne
0
```
Pass by Value: Arrays

Array types are **reference types**, so things work a bit differently with arrays as arguments:

- the array itself (and its length) cannot be changed;
- but its elements can be changed.
- So changing the value of the element of an array is a side-effect of the function.
Pass by Value: Arrays

```java
public class AddOne {
    public static void addOne(int[] anArray) {
        anArray[0]++;
    }
    public static void main(String[] args) {
        int[] a = { 0, 1 };
        addOne(a);
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i]);
        }
    }
}
```

Output

```
% java AddOne
1
1
```
Solution - Using Functions

class NearestNeighbour {
    public static double[] parseArguments(String[] args) {...}
    public static double[] getCentre(double[] points) {...}
    public static void printCentre(double[] centre) {...}
    public static double[] getNeighours(double[] points) {...}
    public static double distance(double x0, double y0,
                                   double x1, double y1) {...}
    public static double[] calcDistances(double[] centre,
                                          double[] neighbours) {...}
    public static void printDistances(double[] dists,
                                       double[] neighbours) {...}
    public static double calcMinimum(double[] dists) {...}
    public static void printMinimum(double min) {...}

    public static void main(String[] args) {
        double[] points = parseArguments(args);
        double[] centre = getCentre(points);
        printCentre(centre);
        double[] neighbours = getNeighours(points);
        double[] distances = calcDistances(centre, neighbours);
        printDistances(distances, neighbours);
        double minimum = calcMinimum(distances);
        printMinimum(minimum);
    }
}
Signature

The **signature** of a Java function consists of its name and its parameter list (number and type of parameters, in order).

**Example signature**

```java
max(int x, int y)
```
The signature of a Java function consists of its name and its parameter list (number and type of parameters, in order).

Example signature

max(int x, int y)

However, it’s often convenient to use the term more loosely to refer to the head of the function definition:

Example head of definition

public static int max(int x, int y)
Return type of a function is stated in the header of the function declaration.

- A function declared `void` doesn't return a value.
- Any function with a non-void return type `rtype` must contain a statement of the form

  ```
  return returnValue;
  ```

where the data type of `returnValue` matches the type `rtype`. 
Cubes, 1

Cubes, version 1

```java
public class Cubes1 {
    public static int cube(int i) {
        int j = i * i * i;
        return j;
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i <= n; i++) {
            System.out.println(i + " " + cube(i));
        }
    }
}
```
```java
public class Cubes1 {
    public static int cube(int i) {
        int j = i * i * i;
        return j;
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i <= n; i++) {
            System.out.println(i + " " + cube(i));
        }
    }
}
```

Output:
```
% java Cubes1 6
0 0
1 1
2 8
3 27
4 64
5 125
6 216
```
public class Cubes2 {
    public static int cube(int i) {
        int i = i * i * i;
        return i;
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i <= n; i++) {
            System.out.println(i + " " + cube(i));
        }
    }
}
public class Cubes2 {
    public static int cube(int i) {
        int i = i * i * i;
        return i;
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i <= n; i++) {
            System.out.println(i + " " + cube(i));
        }
    }
}
public class Cubes3 {
    public static int cube(int i) {
        int j = i * i * i;
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i <= n; i++) {
            System.out.println(i + " " + cube(i));
        }
    }
}
public class Cubes3 {
    public static int cube(int i) {
        int j = i * i * i;
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i <= n; i++) {
            System.out.println(i + " " + cube(i));
        }
    }
}
public class Cubes4 {
    public static int cube(int i) {
        i = i * i * i;
        return i;
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i <= n; i++) {
            System.out.println(i + " " + cube(i));
        }
    }
}
public class Cubes4 {
    public static int cube(int i) {
        i = i * i * i;
        return i;
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i <= n; i++) {
            System.out.println(i + " " + cube(i));
        }
    }
}

Don’t do that!
public class Cubes5 {
    public static int cube(int i) {
        return i * i * i;
    }
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i <= n; i++) {
            System.out.println(i + " " + cube(i));
        }
    }
}
What are the Function Bodies for the Distance Problem?

```java
public static void main(String[] args) {
    double[] points = parseArguments(args);
    double[] centre = getCentre(points);
    printCentre(centre);
    double[] neighbours = getNeighbours(points);
    double[] distances = calcDistances(centre, neighbours);
    printDistances(distances, neighbours);
    double minimum = calcMinimum(distances);
    printMinimum(minimum);
}
```
public static double[] parseArguments(String[] args) {
    int N = args.length;
    if (N % 2 != 0) N--;  // ignore final arg if odd number
    double[] p = new double[N];

    for (int i = 0; i < N; i++)
        p[i] = Double.parseDouble(args[i]);

    return p;
}

public static void main(String[] args) {
    double[] points = parseArguments(args);
    ...
}
public static double[] getCentre(double[] points) {
    // first point is our centre
    double[] c = { points[0], points[1] };
    return c;
}

public static void printCentre(double[] centre) {
    System.out.printf("Centre lies at (%5.2f, %5.2f)\n",
        centre[0], centre[1]);
}

public static void main(String[] args) {
    ...
    double[] centre = getCentre(points);
    printCentre(centre);
    ...
}
Neighbours

public static double[] getNeighbours(double[] points) {
    double[] n = new double[points.length - 2];
    // all except the first are neighbours
    for(int i = 2; i < points.length; i++)
        n[i - 2] = points[i];
    return n;
}

public static void main(String[] args) {
    ...;
    double[] neighbours = getNeighbours(points);
    ...;
}
public static double distance(double x0, double y0,
   double x1, double y1) {
    double d1 = x0 - x1;
    double d2 = y0 - y1;
    return Math.sqrt(d1*d1 + d2*d2);
}

public static double[] calcDistances(double[] centre, double[] neighbours) {
    double[] dists = new double[neighbours.length / 2];
    // step over two at a time to get x and y
    for(int i = 0; i < neighbours.length; i += 2)
        dists[i / 2] = distance(centre[0], centre[1],
                                 neighbours[i], neighbours[i + 1]);
    return dists;
}

public static void main(String[] args) {
    ...
    double[] distances = calcDistances(centre, neighbours);
    ...
}
public static void printDistances(double[] dists, double[] neighbours) {
    for(int i = 0; i < dists.length; i++)
        System.out.printf("Distance to (%5.2f, %5.2f) is %5.2f\n", 
                        neighbours[(i*2)], neighbours[(i*2) + 1], dists[i]);
}

public static void main(String[] args) {
    ...
    printDistances(distances, neighbours);
    ...
}
public static double calcMinimum(double[] dists) {
    double min = dists[0];
    for (int i = 1; i < dists.length; i++)
        if (dists[i] < min) min = dists[i];
    return min;
}

public static void printMinimum(double min) {
    System.out.printf("Minimum distance to " +
            "centre is %5.2f\n", min);
}

public static void main(String[] args) {
    ...
    double minimum = calcMinimum(distances);
    printMinimum(minimum);
}
Functions and Modularity

Advantages of breaking a program into functions:

- decomposition of a complex programming task into simpler steps
- reducing duplication of code within a program
- enabling reuse of code across multiple programs
- hiding implementation details from callers of the function, hence
- readability, via well-chosen names.

Whenever you can clearly separate tasks within programs, you should do so. Aim for methods of no more than 10-15 lines. Shorter is often good.
public class Duplication0 {
    public static void main(String[] args) {
        String boyFirstName = "Jock";
        String boySecondName = "McIness";
        String boyName = boyFirstName + " " + boySecondName;
        int boyWeeklyPocketMoney = 2;
        int boySavingsTarget = 10;
        int boyWeeksToTarget = boySavingsTarget / boyWeeklyPocketMoney;
        System.out.print(boyName + " needs to save for ");
        System.out.println(boyWeeksToTarget + " weeks");

        String girlFirstName = "Jane";
        String girlSecondName = "Andrews";
        String girlName = girlFirstName + " " + girlSecondName;
        int girlWeeklyPocketMoney = 3;
        int girlSavingsTarget = 9;
        int girlWeeksToTarget = girlSavingsTarget / girlWeeklyPocketMoney;
        System.out.print(girlName + " needs to save for ");
        System.out.println(girlWeeksToTarget + " weeks");
    }
}
Code Duplication and Refactoring

Output

% java Duplication0
Jock McIness needs to save for 5 weeks
Jane Andrews needs to save for 3 weeks
public class Duplication1 {

  public static String joinNames(String n1, String n2){
      return n1 + " " + n2;
  }

  public static void main(String[] args) {
      String boyName = joinNames("Jock", "McInnes");
      int boyWeeklyPocketMoney = 2;
      int boySavingsTarget = 10;
      int boyWeeksToTarget = boySavingsTarget / boyWeeklyPocketMoney;
      System.out.print(boyName + " needs to save for ");
      System.out.println(boyWeeksToTarget + " weeks");

      String girlName = joinNames("Jane", "Andrews");
      int girlWeeklyPocketMoney = 3;
      int girlSavingsTarget = 9;
      int girlWeeksToTarget = girlSavingsTarget / girlWeeklyPocketMoney;
      System.out.print(girlName + " needs to save for ");
      System.out.println(girlWeeksToTarget + " weeks");
  }
}
public class Duplication2 {

    public static String joinNames(String n1, String n2){
        return n1 + " " + n2;
    }

    public static int weeksToSavePocketMoney(int pocketMoney, int savingsTarget){
        return savingsTarget / pocketMoney;
    }

    public static void main(String[] args) {
        String boyName = joinNames("Jock", "McInnes");
        int boyWeeksToTarget = weeksToSavePocketMoney(2, 10);
        System.out.print(boyName + " needs to save for ");
        System.out.println(boyWeeksToTarget + " weeks");

        String girlName = joinNames("Jane", "Andrews");
        int girlWeeksToTarget = weeksToSavePocketMoney(3, 9);
        System.out.print(girlName + " needs to save for ");
        System.out.println(girlWeeksToTarget + " weeks");
    }
}
public class Duplication3 {

    public static String joinNames(String n1, String n2){
        return n1 + " " + n2;
    }

    public static int weeksToSavePocketMoney(int pocketMoney, int savingsTarget){
        return savingsTarget / pocketMoney;
    }

    public static void printWeeksToSave(String name, int target){
        System.out.print(name + " needs to save for ");
        System.out.println(target + " weeks");
    }

    public static void main(String[] args) {
        String boyName = joinNames("Jock", "McInnes");
        printWeeksToSave(boyName, weeksToSavePocketMoney(2, 10));
        String girlName = joinNames("Jane", "Andrews");
        printWeeksToSave(girlName, weeksToSavePocketMoney(3, 9));
    }
}
Q: What is ‘refactoring’?

A: The process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure (M. Fowler).

Extract Method is only one of 70 refactoring techniques listed by Fowler at http://www.refactoring.com/.
Q: What is ‘refactoring’?

A: the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure (M. Fowler)
Q: What is ‘refactoring’?
A: the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure (M. Fowler)

Extract Method is only one of > 70 refactoring techniques listed by Fowler at http://www.refactoring.com/.
public class Duplication4 {
    public static String joinNames(String n1, String n2) {
        String title;
        if (n1 == "Jock") title = "Master";
        else title = "Miss";
        return title + " " + n1 + " " + n2;
    }

    public static int weeksToSavePocketMoney(int pocketMoney, int savingsTarget) {
        double sweeties = 0.25;
        double reducedPocketMoney = pocketMoney * (1 - sweeties);
        return (int) (savingsTarget / reducedPocketMoney);
    }

    public static void printWeeksToSave(String name, int target) {
        System.out.println();
        System.out.println("***********************************************");
        System.out.println(name + " needs to save for " + target + " weeks");
    }

    public static void main(String[] args) {
        String boyName = joinNames("Jock", "McInnes");
        printWeeksToSave(boyName, weeksToSavePocketMoney(2, 10));

        String girlName = joinNames("Jane", "Andrews");
        printWeeksToSave(girlName, weeksToSavePocketMoney(3, 9));
    }
}

Functionality has changed, so not a case of refactoring:

**Output**

% java Duplication4

******************************************************************************
Master Jock McInnes needs to save for 6 weeks

******************************************************************************
Miss Jane Andrews needs to save for 4 weeks

But wrapping code up in functions makes it much easier to localize modifications.
Summary: Using Functions / Static Methods

Java functions:
▶ Take zero or more input arguments.
▶ Return at most one output value.
▶ Can have side effects; e.g., send output to the terminal.
Structuring your code with methods has the following benefits:

- encourages good coding practices by emphasizing discrete, reusable methods;
- encourages self-documenting code through good organization;
- when descriptive names are used, high-level methods can read more like a narrative, reducing the need for comments;
- reduces code duplication.
What about recursive functions?
  - Basic concepts same as in Haskell.
  - One exercise (factorial) in this week’s labsheets.

Refactoring improves the structure of code without changing the functionality of the application.
Learning Outcomes for this week

▶ Define and call static functions with zero or more parameters.
▶ Pass the return value of one function as the argument of another one.
▶ Declare and use local variables in functions.
▶ Use functions to modify an array ‘in-place’.
▶ Refactor a short program so that functionality is modularized into functions.
Reading

The order of topics in the Java Tutorial is different from the order of these slides, so at this point there isn’t an ideal match: the following reading anticipates some things we’ll cover later.

Java Tutorial
(Re)read pp33-37; then read pp87-99.
i.e., read the first part of Chapter 2 *Object-Oriented Programming Concepts* carefully now, but stop at *Inheritance*; and read the first part of Chapter 4 *Classes and Objects*, stopping at *Objects*. 