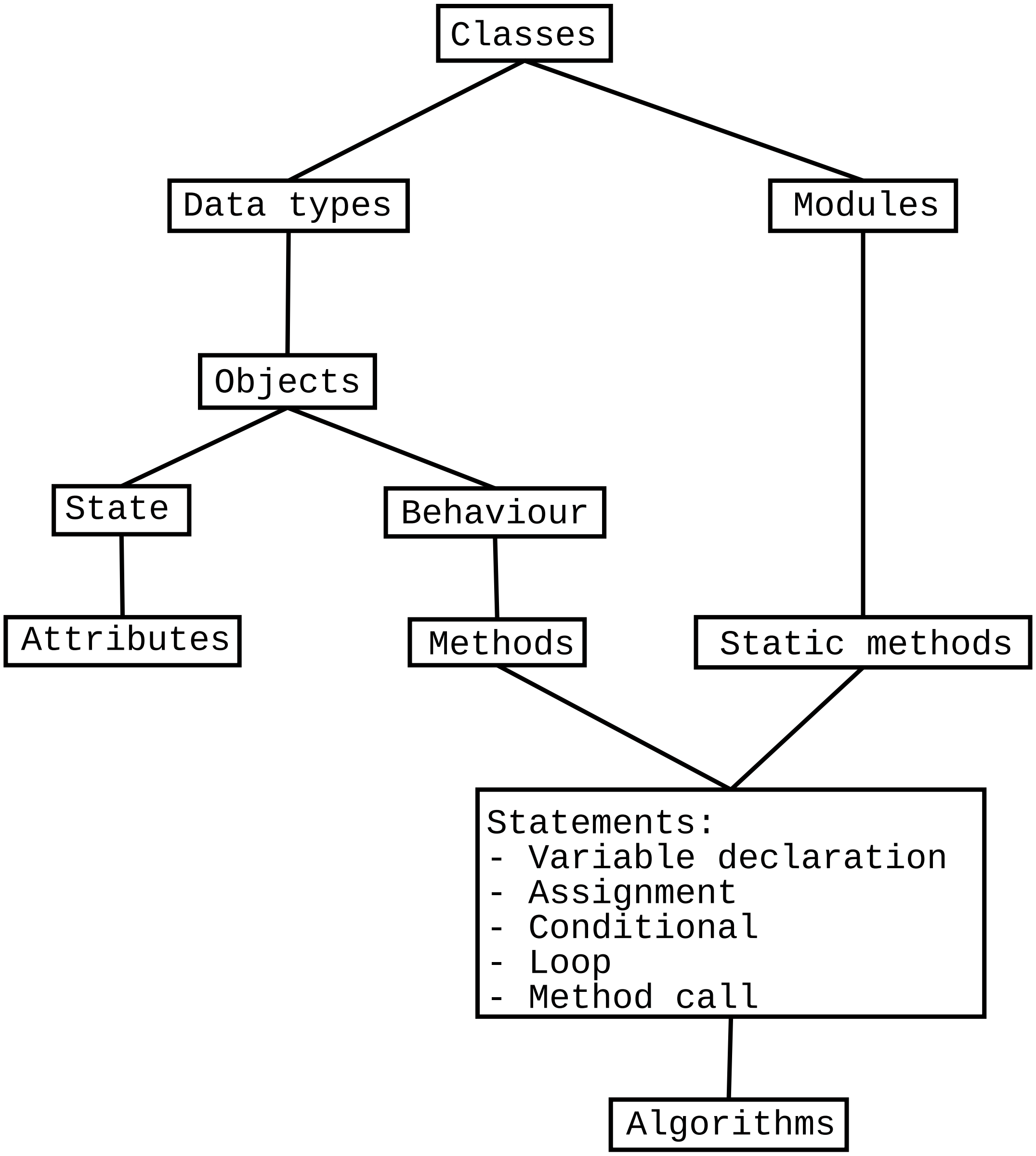
Announcements

- Deadline extended until Thursday, February 26th at 23:55
- Review tutorials: check website for times
- Office hours

Object-Oriented Programming



Static methods

- Normal (non-static) methods represent the behaviour of objects

- Static methods are not associated with objects

- Static methods are only “services” provided by a class

- For example:

  - Keyboard.readString
  - Keyboard.readInt
  - Math.sqrt
  - Math.pow
  - ...etc

Calling normal methods

- When calling a non-static method, the syntax is

  objectreference.method_name(*arg1*,*arg2*,...,*argn*)

  where variable has a reference to an object (e.g. *objectreference* = new *MyClass*();)

  For example:

  ```
  String title = new String(“Lock, Stock”);
  int size = title.length();
  char initial = title.charAt(0);
  ```

Calling static methods

- When calling a static method, the syntax is

  class_name.method_name(*arg1*,*arg2*,...,*argn*)

  Forexample:

```
double power = Math.pow(2.0, 3);
int n = Keyboard.readInt();
```

Declaring methods

- Declaring normal methods

```
type method_name(type1 arg1, type2 arg2,
                 ..., typen argn)
{
    statements;
}
```

- Declaring static methods

```
static type method_name(type1 arg1, type2 arg2,
                        ..., typen argn)
{
    statements;
}
```

Example

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
    }
    static void q()
    {
        System.out.println(“Good bye”);
    }
}
```

(Note: Classes can have both static and non-static methods)

Calling static methods

- A call to a static method takes the form

  ```
  class_name.method(arg1,arg2,...,argn)
  ```

- When the method is called, the corresponding frame does not have a reference to `this`, because there is no object receiving the message.

Example (contd.)

```
public class B
{
    public static void main(String[] args)
    {
        A.q();          // Prints Good bye
        A x = new A();  // Creates an A object
        x.p();          // Prints Hello
        A.p();          // Compile-time Error
        x.q();          // Prints Good bye
    }
}
```

Static methods access

- Since the frame of a static method does not have a reference to an object, static methods cannot access attributes of an object

```
public class A
{
  int n;
  void p()
  {
    System.out.println(n); //OK
  }
  static void q()
  {
    System.out.println(n); //WRONG
  }
}
```

Static methods access

- Since the frame of a static method does not have a reference to an object, static methods cannot access attributes of an object

```
public class A {
  int n;
  void p()
  {
    System.out.println(this.n); //OK
  }
  static void q()
  {
    System.out.println(this.n); //WRONG
  }
}
```

Static methods access

- A static method can be called from a non-static context, but...

- A non-static method cannot be called from a static context, because in order to call a non-static method, you need to provide a reference to an object.

Accessing static methods from non-static methods

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
        q();
    }
    static void q()
    {
        System.out.println(“Good bye”);
    }
}
```

… is OK

Accessing static methods from non-static methods

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
        this.q();
    }
    static void q()
    {
        System.out.println(“Good bye”);
    }
}
```

Accessing static methods from non-static methods

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
        A.q();
    }
    static void q()
    {
        System.out.println(“Good bye”);
    }
}
```

Accessing non-static methods from static methods

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
    }
    static void q()
    {
        System.out.println(“Good bye”);
        p();
    }
}
```

... is **not** OK, because in method q, there is no reference “`this`” to an object to which the message “`p()`” would be sent.

Accessing non-static methods from static methods

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
    }
    static void q()
    {
        System.out.println(“Good bye”);
        this.p();
    }
}
```

When to use each kind of method

- Non-static methods are used to describe the behaviour of objects.
- Static methods are used to describe functions, or services that a class provides, independently of any object of that class.

Example

```
public class Distance
{
  static double euclidean(float x1, float y1,
                          float x2, float y2)
  {
      return Math.sqrt(Math.pow(x1-x2,2)
                       + Math.pow(y1-y2,2));
  }
  static double manhattan(float x1, float y1,
                          float x2, float y2)
  {
      return Math.abs(x1-x2)+Math.abs(y1-y2);
  }
}
```

Example (contd.)

```
import cs1.Keyboard;
public class ComputeDistance
{
    public static void main(String[] args)
    {
        float x1, y1, x2, y2;
        double e, m;
        x1 = Keyboard.readFloat();
        y1 = Keyboard.readFloat();
        x2 = Keyboard.readFloat();
        y2 = Keyboard.readFloat();
        e = Distance.euclidean(x1, y1, x2, y2);
        m = Distance.manhattan(x1, y1, x2, y2);
        System.out.println(“Euclidean: ”+e);
        System.out.println(“Manhattan: ”+m);
    }
}
```

Methods as procedural abstractions

- A method implements an algorithm
- The steps of an algorithm might be complex ...
- ... therefore, its steps can be implemented as separate methods.
- A method abstracts the way in which a particular step, operation, function or algorithm works.
- Top-down software development:
  - Start from a general algorithm first, and
  - Develop the substeps later. Each substep can be implemented as a separate method.

Example: Newton's algorithm for sqrt

- Problem: Given a positive real number x, compute its square root, $\sqrt{x}$

- Analysis:
  - The square root of a positive real number x, is a real number s such that $s^2 = x$
  - The square root of some positive real numbers has an infinite decimal expansion...
  - ...therefore, we can compute only approximations, i.e. compute a number s such that s^2 is "close enough" to x.
  - Two numbers are "close enough" if the absolute value of the difference between them is very small, i.e. smaller than a given tolerance factor.

Example (contd.)

- Algorithm: Input: *x*, *tolerance*; Output: approx $\sqrt{x}$

1. Start with a guess set to 1
2. While the guess is not good enough (i.e. while $guess^2$ is not close to *x* with respect to the tolerance,) repeat:
    (a) Improve the guess
3. Return the final guess

- So there are two main substeps:
    - Determining whether two numbers are close enough
    - Improving a guess

Example (contd.)

- Determining if two values are "close enough" with respect to a tolerance or not:
  - Input: two values *a* and *b* (reals), the tolerance factor (a positive real)
  - Output: a boolean: true if the guess is good enough w.r.t. the tolerance, false otherwise

1. If $|a - b| < tolerance$ return true

2. otherwise, return false

```
static boolean close_enough(double a, double b,
                            double tolerance)
{
    return (Math.abs(a-b) < tolerance);
}
```

Example (contd.)

- Improving the guess
  - Input: the current guess g (a positive real), and x (a positive real)
  - Output: an improved guess (a positive real,) namely: the average between the current guess and the ratio of x and the current guess.

1. Return $\frac{1}{2}(g + \frac{x}{g})$

```
static double improve(double g, double x)
{
    return (g + x/g)/2;
}
```

Example (contd.)

```
public class Newtons {
  static double sqrt(double x, double tolerance)
  {
    double guess = 1.0;
    while (!close_enough(guess*guess,x,tolerance))
    {
      guess = improve(guess, x);
    }
    return guess;
  }
  static boolean close_enough(double a, double b,
                              double tolerance)
  {
    return (Math.abs(a-b) < tolerance);
  }
  static double improve(double g, double x)
  {
    return (g + x/g)/2;
  }
}
```

Example (contd.)

```
public class Distance
{
  static double euclidean(float x1, float y1,
                          float x2, float y2)
  {
      return Newtons.sqrt(Math.pow(x1-x2,2)
                          + Math.pow(y1-y2,2),
                          0.001);
  }
  static double manhattan(float x1, float y1,
                          float x2, float y2)
  {
      return Math.abs(x1-x2)+Math.abs(y1-y2);
  }
}
```

Example (contd.)

```
public class Newtons {
  static double sqrt(double x, double tolerance)
  {
    double guess = 1.0;
    while (!(Math.abs(guess*guess-x) < tolerance))
    {
      guess = (guess + x/guess)/2;
    }
    return guess;
  }
}
```

Searching for solutions

- Generic algorithm to search for solutions:

1. Start with some guess

2. While the guess is not good enough, repeat:

   (a) Improve the guess

3. The result is the final guess

Static methods (contd.)

- Static methods represent procedural abstractions
- Why don't we use only static methods and no non-static methods? We could, but we want to use OOP, because we want to model the problem domain realistically. Objects and classes do that.
- Static methods: functional/procedural view of the problem and its solution.
- Non-static methods: object-oriented view of the problem and its solution.

Methods a *reusable* abstractions

- A method can be reused in different contexts
- Calling a method is “the same” as substituting its body in place of its call (replacing the parameters by the actual arguments,) but
- If we define a method, we can simply call it from more than one context without having to do copy and paste.

Example: reusing methods

```
public class B {
  void q(int v)
  {
    int k = (v+1)*2+1;
    // ... do something with k
  }
}
public class C {
  void r(int w)
  {
    int u = (w-3)*2+1;
    // ... do something with u
  }
}
```

Example (contd.)

```
public class A {
  static int p(int n)
  {
    return n*2+1;
  }
}
public class B {
  void q(int w)
  {
    int k = A.p(w+1);
    // ... do something with k
  }
}
public class C {
  void r(int v)
  {
    int u = A.p(v-3);
    // ... do something with u
  }
}
```

Static variables

- The attributes of a class are normal variables.
- The values of these attributes are individual to each object in a class.

```
public class A {
  int x;
}
public class B {
  void m()
  {
    A u = new A();
    A v = new A();
    u.x = 5;
    v.x = -7;
    // Here, u.x == 5 and v.x == -7
  }
}
```

Static variables (contd.)

- Static variables are attributes of the class, not of the objects

- Static variables are shared between all the objects in a class

```
public class A {
  static int x;
}
public class B {
  void m()
  {
    A u = new A();
    A v = new A();
    u.x = 5;
    v.x = -7;
    // Here, u.x == -7 and v.x == -7
  }
}
```

Static variables (contd.)

```
public class BankAccount
{
  float balance;

  BankAccount()
  {
    balance = 0.0f;
  }
  void deposit(float amount)
  {
    balance = balance + amount;
  }
  void withdraw(float amount)
  {
    if (amount < balance)
      balance = balance - amount;
  }
}
```

Static variables (contd.)

```
public class Bank {
  public static void main(String[] args)
  {
    BankAccount pete, amy;
    pete = new BankAccount();
    amy = new BankAccount();

    pete.deposit(700.0f);
    amy.deposit(800.0f);

    System.out.println(pete.balance);
    System.out.println(amy.balance);
  }
}
```

Static variables (contd.)

```
public class BankAccount
{
  static float balance;

  BankAccount()
  {
    balance = 0.0f;
  }
  void deposit(float amount)
  {
    balance = balance + amount;
  }
  void withdraw(float amount)
  {
    if (amount < balance)
      balance = balance - amount;
  }
}
```

Static variables (contd.)

```
public class Bank {
  public static void main(String[] args)
  {
    BankAccount pete, amy;
    pete = new BankAccount();
    amy = new BankAccount();

    pete.deposit(700.0f);
    amy.deposit(800.0f);

    System.out.println(pete.balance);
    System.out.println(amy.balance);
  }
}
```

The end