UCT Department of Computer Science

Computer Science 1015F

Object Oriented Programming

Hussein Suleman
<hussein@cs.uct.ac.za>
March 2009

Objects

- Objects are computer representations of real-world or abstract objects.
  - e.g., input, System.out, aPerson, timTheTurtle
- Objects are modelled on computer as complex data types, defining possibly multiple values AND various operations that may be applied to those values.
- This style of programming is called Object Oriented Programming (OOP).
- Why OOP?

Classes

- Classes are **templates** to create objects.
- Classes define the **data** and associated operations (**methods**) for objects of a particular type.

```
public class ClassName
{
   // data and methods here
}
```

- A class is a type, just like int, boolean, etc.
- One class in every file must be public - exposed to the outside.
- Separate files = modular programming

Instances

- An **instance** is a variable of the type corresponding to a particular class.
- Instances are often simply called **objects**.
- Unlike variables with primitive types (e.g., int), instances are not created when the variable is declared.
- To create an instance from a class use **new**
- Simplified syntax:
  - `<class_name> <variable name>;`
  - `<variable name> = new <class_name> ();`
  - Examples:
    - `Person aPerson;`
    - `aPerson = new Person ();`

Instance variables

- **Instance variables** are variables defined within a class, with separate copies for each instance.
- This makes every object unique, even though they have the same class.
  - Just like different int variables are unique but all have the same type!
- Instance variables are usually labelled **private** because they may only be used by methods within this class.

```
public class Person
{
   private String firstName, lastName;
   private int age;
}
```

Methods

- A **method** is a block of statements within a class.
- It is considered a single unit, and named with an identifier.
  - Just like a variable.
- It is used for common functions/subprograms and to set/retrieve values of instance variables from outside the object.
- A method is **called** or **invoked** using **dot-notation** in the context of an object.
  - e.g., `System.out.println (“Hello”);`
  - System.out is the object. println is the method executed on that object.

- When a method is called, execution jumps to the method and only comes back when the method is finished.

Methods: Data In

- **Parameters** are used to send data to a method - within the method they behave just like variables.

```
public void setName ( String first, String last )
{
   firstName = first; lastName=last;
}
```

- Calling methods must provide matching values (**arguments**) for every parameter.
  - e.g., `aPerson.setName (“Alfred”, “Tshabalala”);`
- Formal parameters (first) vs. Actual parameters (“Alfred”)

Methods: Data Out

- Values can be returned from a **typed method**.

```
public int getAge ()
{
    return age;
}
```

- **return** must be followed by an expression with the same type as the header (int in above example).

- So what is an **untyped method**?
  - One whose type is indicated as **void**.
- return can be used to simply leave the method.

Method Syntax

- Simplified syntax:

```
public <type> <method_name> (<list_of_parameters>)
{
   <list_of_statements>
}
```

- Example:

```
public int doAdd ( int aValue, int anotherValue )
{
   int sum = aValue+anotherValue;
   return sum;
}
```

Methods: Quick Quiz

```
public class Planet {
   private String name;
   public void setName ( String aName ) {
      name = aName;
   }
}
...
Planet earth = new Planet ();
```

- Which of these work?
  - `earth.setName ();`
  - `earth.setName (2.345);`
  - `earth.setName (“Mars”);`
  - `earth.setName (“Mercury”, “Venus”, “Earth”);`
  - `earth.setName (“The”+“ Dude’s ”+“Planet”);`

Problem

- Write a class that represents complex numbers.
- Use this class to perform simple arithmetic on complex numbers.

Methods to factor common code

```
...
System.out.println (“YAY it works”);
System.out.println (“a=“+a);
...
System.out.println (“YAY it works”);
System.out.println (“a=“+a);
...
System.out.println (“YAY it works”);
System.out.println (“a=“+a);
```

```
...
public void yay ()
{
   System.out.println (“YAY it works);
   System.out.println (“a=“+a);
}
...
   d.yay ();
   d.yay ();
   d.yay ();
```

Methods with parameters

```
…
System.out.println (“YAY it works”);
System.out.println (“a=“+12);
…
System.out.println (“YAY it works”);
System.out.println (“a=“+13);
…
System.out.println (“YAY it works”);
System.out.println (“a=“+14);
```

```
public void yay ( int someNumber )
{
   System.out.println (“YAY it works);
   System.out.println (“a=“+someNumber);
}
…
x.yay (12);
x.yay (13);
x.yay (14);
```

Methods with parameters/return values

```
...
c=a*a+2*a*b+b*b;
...
d=e*e+2*e*f+f*f;
...
g=h*h+2*h*i+i*i;
```

```
public int doCalc ( int n1, int n2 )
{
   return (n1*n1+2*n1*n2+n2*n2);
}
…
c = x.doCalc (a, b);
d = x.doCalc (e, f);
g = x.doCalc (h, i);
```

Local and Instance Variables

- **Local variables** are defined within a method or block (i.e., { and }). Local variables can even be defined in a *for* statement.
  - e.g., for (int a=1; a<10; a++)
- **Instance variables** are defined within a class, but outside any methods, and each object has its own copy.
- A variable has **scope** when it can be used and **lifetime** when it exists.

this

- *this* is a special instance variable that exists in every instance.
- *this* refers to the current object.
- Calling `this.someMethod()` is the same as calling `someMethod()`.
- What is the point of *this*?

equals and toString

- *equals* is a special method with a single parameter being of the same type, returning whether or not the two objects are equal.

```
public boolean equals ( Person aPerson )
{
   return this.name.equals (aPerson.name);
}
```

- *toString* is a special method with no parameters that returns a String representation of the object.

```
public String toString ()
{
   return (name+” ”+surname);
}
```

Problem

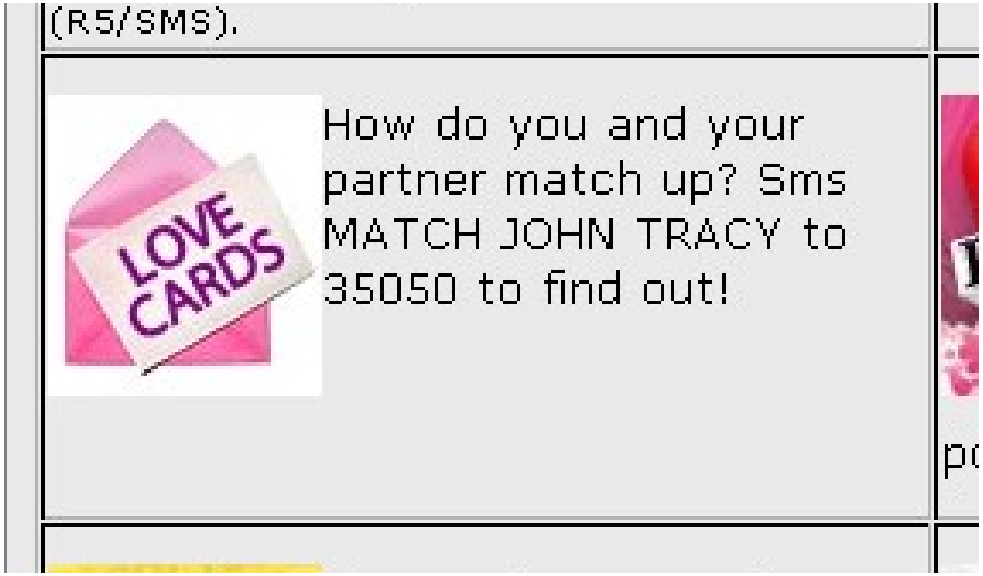
- Write a program to calculate the roots of a quadratic polynomial.

Problem

- Write a program to calculate whether or not a student will get DP and can write the examination in CSC1015F.

Problem

- Write a numerology calculator using object-oriented programming. For any two given birthdates, calculate the compatibility between people as a simple 0-100 integer.
- Use any formula that makes sense.



(R5/SMS).

How do you and your partner match up? Sms MATCH JOHN TRACY to 35050 to find out!

Overloading

- **Overloading** means having multiple methods with the same name and different parameter lists (but same return type) within a single class.

```
class Averages
{
   public int average ( int x, int y )
   {
      return (x + y)/2;
   }
   public int average ( int a, int b,
                        int c )
   {
       return (a + b + c)/3;
   }
}
```

```
Averages ave;
ave = new
Averages();

int a = ave.average
         (1,2);
int b = ave.average
         (1,2,3);
```

Why overload?

- A programmer using the class can use the same method name for different parameters if the name is sensible.

- Remove the need for lots of unique names for methods that essentially do the same thing.

Problem

- Modify the Complex number class to use overloading to avoid multiple multiplication methods.

Constructors

- An object is initialised (given initial values) by means of a special method called a **constructor**.

- Every class may have one or more of these special methods with no return type and the same name as the class.

```
public class Person
{
   public Person ( String firstname )
   { … }
}
Person aPerson = new Person (“hussein”);
```

Initialising Objects with Constructors

- Create an object using **new** operator followed by the name of the class and the parameters/arguments to a constructor.
- Constructors can be overloaded.
  - Normally include a constructor with no arguments so you can say:
    - `Person aPerson = new Person();`
- Constructors cannot be invoked directly.

Problem

- Write a OO program to calculate some basic statistics for a class test – including average, minimum and maximum marks (and track the names of best/worst students).

Problem

- Add suitable constructors to the Complex number class.

Other ways to initialise objects

- Assume variables are initialised to “zero”. Java does this automatically for primitive instance variables!

- Initialise instance variables in the class definition.

```
public Person
{
   String firstname = “John”;
   String lastname = “”;
   public Person ( String fname, String lname )
   {… }
```

StringTokenizer

- Class to separate a String into multiple words.
- Typical Use:

```
String as = “Hello World”;
StringTokenizer st = new StringTokenizer (as);
while (st.hasMoreTokens())
{
   System.out.println (st.nextToken());
}
```

Encapsulation

- **Encapsulation** in Java is the combining of data and methods into single units.

- This allows us to treat the object as a single unit, preventing errors when keeping track of multiple related variables and methods.

Information Hiding

- **Information hiding** means we don't allow programmers to see details that they don't need to see.
- This means fewer accidental programming errors.
- Java enables this with the *public* and *private* prefixes/ modifiers.

public and private

	instance variable	method
public	accessible from anywhere `public int x;`	accessible from anywhere `public int getAge ();`
private	accessible from methods in same class `private int x;`	accessible from methods in same class `private int getAge();`

Accessors and Mutators

- **Accessors** are methods that allow you to access one (or more) private instance variable(s).

```
public int getAge ()
{
    return age;
}
```

- **Mutators** are methods that allow you to set the value of one (or more) private variable(s).

```
public void setAge ( int anAge )
{
   age = anAge;
}
```

Why accessors and mutators?

- Control access to instance variables by providing only some accessors and mutators = information hiding.

- Allow additional sanity checks when assigning values for instance variables.
  - e.g., check that a date is valid