ESc101 : Fundamental of Computing

I Semester 2008-09

Lecture 22

Object Oriented programming

- Building complex classes using simple classes
- Packages (library of related classes)
- Example with Object reference as parameters in Methods
- Access controls (public, default(package), private)
- Method Overloading in Java.

A common confusion : what is x in distance_from_origin ?

```
public class Point
{   double x;
    double y;

    public Point(double x1, double y1)
    {   x = x1;
        y = y1;
    }

    public double distance_from_origin()
    {   double dist;
        dist = Math.sqrt(x*x + y*y);
        return dist;
    }
}
```

Resolving the confusion using this reference

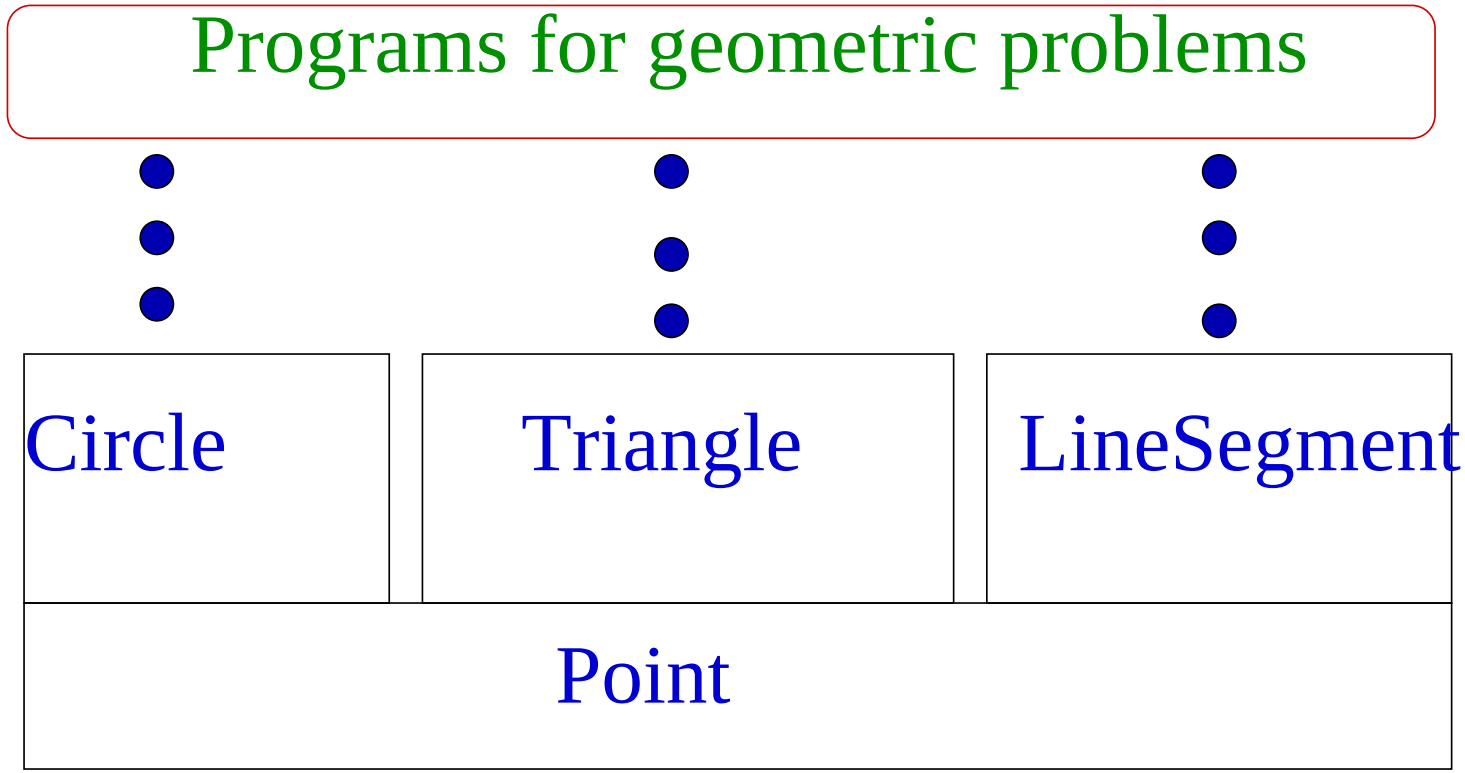
```
public class Point
{   double x;
    double y;

    public Point(double x1, double y1)
    {   this.x = x1;
        this.y = y1;
    }

    public double distance_from_origin()
    {   double dist;
        dist = Math.sqrt(this.x*this.x + this.y*this.y);
        return dist;
    }
}
```

this refers to the current object being accessed

Advantages of taking OOP



Organizing a collection of related classes

- It would be very convenient if we keep **Point.java**, **Triangle.java**, **Circle.java** together in one library.
- this library should be imported to any program which requires any class from that library

In JAVA, it is achieved using **Package**

.

Package : library of related classes

Example :

- Keep **Point.java**, **Triangle.java**, and **Circle.java** in a directory.
- Give some name to the directory, say **Geometry**
- In the beginning of all the files/classes in **Geometry**,

  write **package Geometry;** at the top.
- For all programs which require ***class_a*** from **Geometry**,

  write **import Geometry.*class_a*;** at the top.

Home work :

- Include the methods **Area()** of triangle_program1.java in the class Triangle itself.

  Rewrite program for problem 1 based on the modified Triangle class

- Include the method **Intersect_circles()** of circle_program1.java in the class Circle itself. Rewrite program for problem 1 based on the modified Circle class

What will be the output of the following program ?

```
class test1
{
    public static void Increment(int i)
    {
        i = i+1;
    }

    public static void main(String args[])
    {
        int j = 1;
        Increment(j);
        System.out.println(j);
    }
}
```

Output : **?**

What will be the output of the following program ?

```
class test1
{
    public static void Increment(int i)
    {
         i = i+1;
    }

    public static void main(String args[])
    {
        int j = 1;
        Increment(j);
        System.out.println(j);
    }
}
```

Output : **1**

Consider the folowing class

```
public class myInt
{
    int value;
    myInt(int i)
    {  value = i;    }
}
```

What will be the output of the following program ?

```
class test2
{   public static void Increment(myInt i)
    {
        i.value = i.value+1;
    }

    public static void main(String args[])
    {
        myInt j = new myInt(1);
        Increment(j);
        System.out.println(j.value);
    }
}
```

Output : **??**

What will be the output of the following program ?

```
class test2
{   public static void Increment(myInt i)
    {
        i.value = i.value+1;
    }

    public static void main(String args[])
    {
        myInt j = new myInt(1);
        Increment(j);
        System.out.println(j.value);
    }
}
```

Output : **2**

An important rule in JAVA you should never miss

parameter passing in methods always follow **“Call by value”** rule

Members of a class

1. **Attributes or Fields** :

   The data variables associated with a class and its object

2. **Methods**

Access Control of members of a class

- **private** :

  members declared private are accessible only in the class itself.

- **public** :

  members declared public are accessible anywhere the class is accessible.

- **package** :

  members declared with no access modifier are accessible in the classes in the same package.

- **protected** : **perhaps not to be discussed in the course**

General rules (but you may break them sometimes)

1. The attributes should **NEVER be declared public.**

   Advantage : No attribute is modified accidentally by some programmer.

2. **The constructors and other methods should be declared public.**

How to read attributes which are declared private ?

Question : How to read **x** or **y** coordinates of a point object if these attributes in Point class are declared **private**?

Answer : We may define public methods which return attributes.

Example :

```
public class Point
{   private double x;    private double y;

    public double getX()
    {   return this.x;    }

    public double getY()
    {   return this.y;    }

    public Point(double x1, double y1)
    {   this.x = x1;
        this.y = y1;
    }

    public double distance_from_origin()
    {   double dist;
        dist = Math.sqrt(this.x*this.x + this.y*this.y);
        return dist;       }}
```

Method overloading in JAVA

Question : In a class, can there be two methods with the same name?

Method overloading in JAVA

Question : In a class, can there be two methods with the same name?

Yes, but they should differ in the their **signatures**

Signature of method defined in lecture 22

return_type method_name (type1, type2, ...)

For example,

- public static int absolute(int i) has signature

  int absolute(int)

- public static double absolute(int i, double d) has signature

  double absolute (int, double)

Incorrect : return_type is not part of the signature

Signature of method correctly defined is

method_name (type1, type2, ...)

For example,

- public static int absolute(int i) has signature

  absolute(int)

- public static double absolute(int i, double d) has signature

  absolute (int, double)

which method will be executed if the method is overloaded

Suppose there are five methods

1. public static double absolute(int i, int j)
2. public static double absolute(double d, int i)
3. public static double absolute(int i, double d)
4. public static double absolute(double i, double d)
5. public static double absolute(short i, double d)

If we invoke **doble d = absolute(4, 3.5*2)**,

which method will execute ?

which method will be executed if the method is overloaded

Suppose there are five methods

1. public static double absolute(int i, int j)
2. public static double absolute(double d, int i)
3. public static double absolute(int i, double d)
4. public static double absolute(double i, double d)
5. public static double absolute(short i, double d)

If we invoke **double d = absolute(4, 3.5*2)**,

3rd method will execute.

which method will be executed if the method is overloaded

Suppose there are five methods

1. public static double absolute(int i, int j)
2. public static double absolute(double d, int i)
3. public static double absolute(int i, double d)
4. public static double absolute(double i, double d)
5. public static double absolute(short i, double d)

If we invoke **double d = absolute(short(4), 3.5*2)**,

which method will execute ?

which method will be executed if the method is overloaded

Suppose there are five methods

1. public static double absolute(int i, int j)
2. public static double absolute(double d, int i)
3. public static double absolute(int i, double d)
4. public static double absolute(double i, double d)
5. public static double absolute(short i, double d)

If we invoke **double d = absolute(short(4), 3.5*2)**,

5th method will execute ?

which method will be executed if the method is overloaded

Suppose there are five methods

1. public static double absolute(int i, int j)
2. public static double absolute(double d, int i)
3. public static double absolute(int i, double d)
4. public static double absolute(double i, double d)
5. public static double absolute(short i, double d)

If we invoke **double d = absolute(short(4), 3)**,

which method will execute ?

which method will be executed if the method is overloaded

Suppose there are five methods

1. public static double absolute(int i, int j)
2. public static double absolute(double d, int i)
3. public static double absolute(int i, double d)
4. public static double absolute(double i, double d)
5. public static double absolute(short i, double d)

If we invoke **double d = absolute(short(4), 3)**,

Compiler error : reference to absolute is ambiguous since there are two potential candidates for it : **1st** and **5th**.

which method will be executed if the method is overloaded

Suppose there are only Three methods

1. public static double absolute(int i, int j)
2. public static double absolute(int i, double d)
3. public static double absolute(double i, double d)

If we invoke **double d = absolute(short(4), 3)**,

which method will execute ?

which method will be executed if the method is overloaded

Suppose there are only Three methods

1. public static double absolute(int i, int j)
2. public static double absolute(int i, double d)
3. public static double absolute(double i, double d)

If we invoke **double d = absolute(short(4), 3)**,

1st method will be executed since it has at least one match in parameter type whereas **2nd** and **3rd** have no match.

Useful progamming advice

Invoke methods with arguments which **match exactly** the signature of a method.

Do it by explicit type cast of arguments if needed.

What is expected from the students : they should be able to find out the method which will be executed if the types of the arguments **exactly match** some method.