

FIT ICT Software Development

Lecture 9

Lecturer : Charles Tyner

Object Orientation in Java

- **Object Oriented Programming (OOP)**
- **Forces that favour OOP**
- **Declaring and Creating Objects**
 - **Declaring classes**
 - **Creating objects**
 - **Manipulating objects**
- **Information Hiding**
- **The class as the unit of programming in Java**
- **The Constructor Method**
- **Java an extensible language**
- **Software Reuse**
- **Inheritance.**
- **composition and aggregation**

Object Orientation in Java

This session we investigate the topic of **object orientation** in Java.

The **objects** we build will be composed in part of structured program pieces, so we need to establish a basis in structured programming with control structures.

Methods are a part of this, and we have studied, and used, methods in detail.

We have already introduced many of the basic concepts and terminology of object-oriented programming in Java.

We have:

- analyzed many typical problems that require a program (applet or application),
- determined what classes from the Java API were needed to implement the program,
- determined what instance variables our applet or application needed
- determined what methods our applet or application needed to have and
- specified how an object of our class needed to collaborate with objects of Java API classes to accomplish the overall goals of the program.

Programming in Java involves thinking in terms of objects.

A Java program can be viewed as a collection of co-operating objects.

In this lecture we introduce the fundamentals of object-oriented programming, including:

- Declaring classes
- Creating objects
- Manipulating objects

“Programming in the small”

Single programmer understands entire program

Program performs a small pre-specified range of tasks

Programs often take input, perform calculations, return results and stop

e.g. program that takes in an amount, an interest rate, and a time period and calculates the repayment amount on a mortgage

Such smallish programs are often best designed using
... **structured programming**

"Programming in the large"

Most computer programs these days:

- run continuously
- respond to input from users and other programs
- deliver output to users and other programs
- are programmed by large teams of programmers
- are too big and complex for any one person to understand

Real-world Organisation

How is something moderately complex, like a restaurant organised?

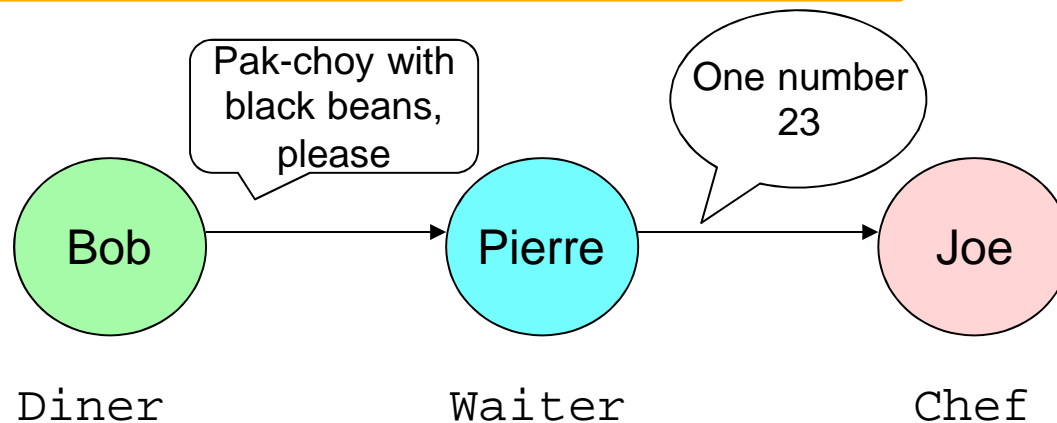
- Diners want meals
- Chefs prepare the dishes
- Waiters take orders from diners, and bring the food to the table
- Busboys collect and wash plates
- Barmen prepare and serve drinks
- The maitre'd makes reservations and seats diners

Each type of person provides a narrow range of services. The restaurant operates through the *co-operative interaction* of all the restaurant staff

Object Oriented Programming

OOP refers to the process of designing and implementing a co-operative community of interacting *objects*.

- each object provides a small number of relatively simple *services*
- objects *communicate* with each other to exchange information more complex behaviour is provided by the *co-operation* of objects



Why OOP?

- Complexity management
 - **encapsulation** and **information hiding** work on the principle that objects should interact *only* through pre-specified **interfaces**
 - software development can be more reliably divided between independent groups
- Reusable software
 - **class libraries** provide easy access to many standard services
 - developing **software components** that match the reliability and interchangeability of hardware components is an elusive goal
- Natural modelling
 - problem identification, program design and program implementation all follow same process

External forces favouring OOP

- Client-server paradigm
 - most modern computing is based on **servers** providing services to **clients** so this extends naturally to programs
- Event driven programming
 - actions occur in response to external events, with the program being otherwise idle
- Increasing use of simulation
 - simulation of physical, natural or human situations deals with complex interactions of large number of relatively simple “atoms”
- Easily adaptable to parallel computing
 - two or more CPUs can operate on the same collection of agents with almost no behavioural change

Objects and Classes

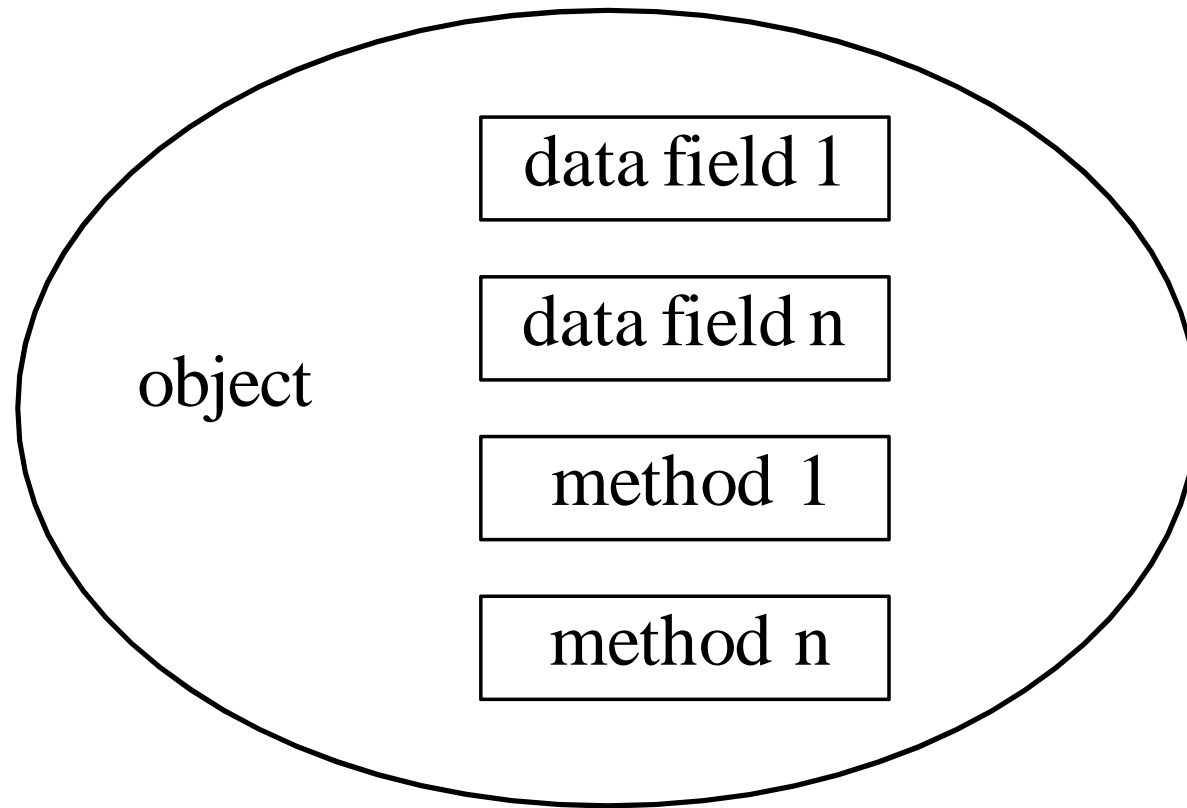
Object-oriented programming involves programming using objects.

Object is a broad term that stands for many things.

A student, a desk, a circle, an Airbus Plane, and even a mortgage loan can all be viewed as objects.

Certain **properties** define an object, and certain **behaviours** define what it does.

These **properties** are known as **data fields**, and the objects **behaviours** are defined by **methods**.

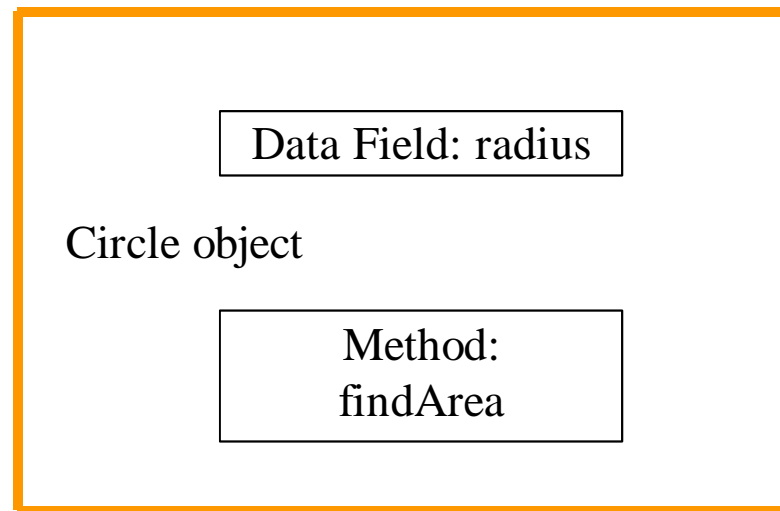


An object contains data and methods.

A Circle object has a data field **radius**.

This is the **property** that characterises a circle.

One **behaviour** of a circle is that its **area** can be computed.



A **Circle** object contains the **radius** data field and the **findArea** method.

Classes are structures that define objects.

In a Java class,

- **data** are used to describe properties and
- **methods** to define behaviours.

A class for an object contains a collection of method and data definitions.

The following is an example of the class for a circle:

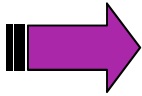
```
class Circle
{
    double radius = 1.0;

    double findArea()
    {
        return radius*radius*3.14159;
    }
}
```

This class is different from all of the other classes that we have seen so far.

The class does not have a **main** method.

Therefore, you cannot run this class.



It is mainly a definition used to declare and create Circle objects.

For convenience, the **class** that contains the **main()** method will be referred to as the **main class**.

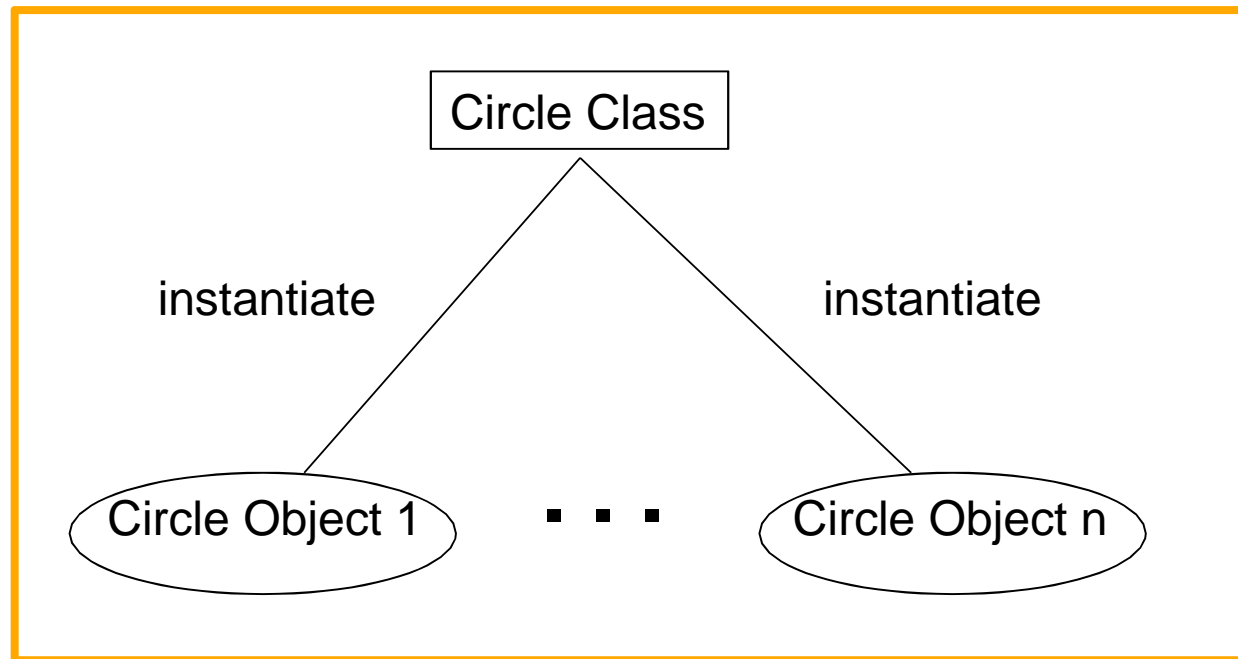
Declaring and Creating objects

A class is a blueprint that defines what an object's data and methods will be.

An object is an instance of a class.

You can create many instances of a class.

A class can have many different objects



The relationship between classes and objects is analogous to the one between apple pie recipes and apple pies.

You can make as many apple pies as you like from a single recipe.

Creating an instance is referred to as **instantiation**.

In order to declare an object, you must use a variable to represent it.

In this regard, it is similar to defining a variable for a primitive data type.

The syntax for declaring an object is as follows:

```
ClassName objectName;
```

The following statement declares the variable `myCircle` to be an instance of the `Circle` class:

```
Circle myCircle;
```

Creating an object of a class is called **creating an instance of the class**.

An object is a variable that has a class type.

To create variables of a primitive data type, you would simply declare them, as we do in the line following:

```
int k;
```

This statement of Java creates a variable and allocates memory space for it.

For object variables, however, declaring and creating are two separate steps.

- The declaration of an object simply associates the object with a class, making it an instance of that class.
- The declaration does not create the object.

To actually create **myCircle**, you must use the operator **new** in order to tell Java to create an object for **myCircle** and allocate memory for it.

The syntax for creating an object is as follows:

```
objectName = new ClassName( );
```

For example, the following statement creates an object, myCircle, and allocates memory for it:

```
myCircle = new Circle( );
```

You can combine the declaration and instantiation together in one statement in one step:

```
ClassName objectName = new ClassName( );
```

For our `myCircle` example, this becomes:

```
Circle myCircle = new Circle( );
```

After an object is created, it can access its data and methods by the following notation:

ObjectName.data

References the object's data

ObjectName.method

References the object's method

For example,

myCircle.radius indicates what the radius of myCircle is, and
myCircle.findArea() returns the area of myCircle.

We will see the actual code for this shortly!



TextPad - [C:\ITB\SoftDev-1_JAVA\myPrograms\TestCircle.java]

File Edit Search View Tools Macros Configure Window Help

```
1 class TestCircle
2 {
3     public static void main(String[] args)
4     {
5         Circle myCircle = new Circle();
6         System.out.println("The area of the circle of radius "
7             + myCircle.radius + " is " + myCircle.findArea());
8
9         //pause
10        System.out.println("Press Ctrl+C to close this window ...");
11        MyInput.readInt();
12    }
13 }
14
15 class Circle
16 {
17     double radius = 1.0;
18
19     double findArea()
20     {
21         return radius*radius*3.14159;
22     }
23 }
24
25
26
27
28
29
30
31
```

TestCircle.java

ASCII Characters

| | |
|----|----|
| 33 | ! |
| 34 | " |
| 35 | # |
| 36 | \$ |
| 37 | % |
| 38 | & |
| 39 | ' |
| 40 | (|
| 41 |) |
| 42 | * |
| 43 | , |
| 44 | - |
| 45 | . |
| 46 | : |
| 47 | / |
| 48 | [|

MS-JAVA

Auto

The area of the circle of radius 1.0 is 3.141592653589793
Press Ctrl+C to close this window ...

File: TestCircle.java, 464 bytes, 30 lines, ANSI

1 1 Read Out Block Dyna Text Code 25:23

The program contains two classes.

The first class, **TestCircle**, is the **main class**.

Its sole purpose is to test the second class, **Circle**.

Every time you run the program, the Java runtime system invokes its **main()** method in the main class.

The main class contains the **main()** method that creates an object of the **Circle** class and prints its radius and area.

The **Circle** class contains the **findArea()** method and the radius data field.

In this object-oriented programming world of Java,
... the **radius** and **findArea()** are defined in the same class.

The **radius** is a data member in the **Circle** class which is accessible
to the **findArea()** method.

The **findArea()** method is an **instance** method, which is always
invoked by an **instance** in which the radius is created.

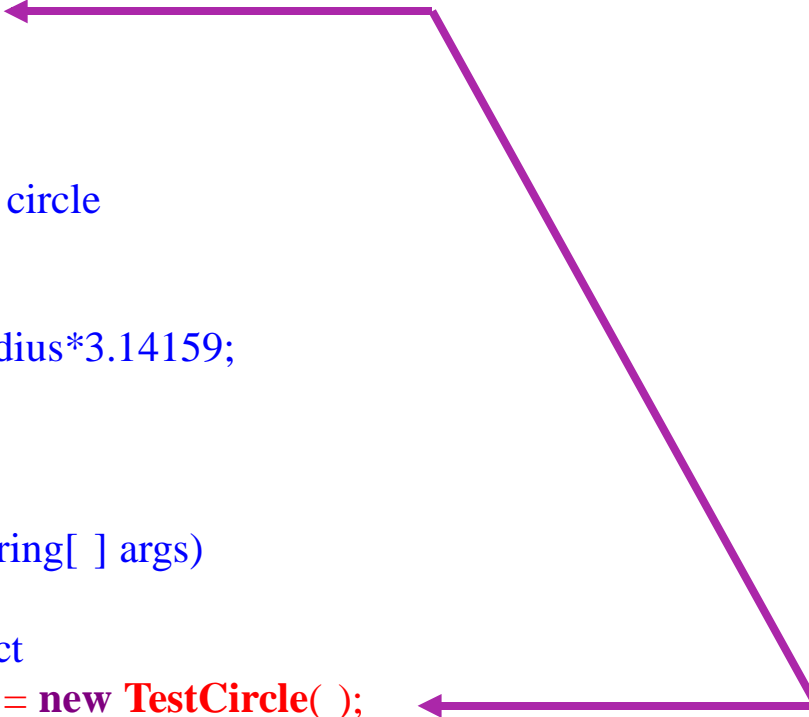
There are many ways to write Java programs.

For instance, you can combine the two classes in the example into one.

```
public class TestCircle
{
    double radius = 1.0;

    //find the area of this circle
    double findArea( )
    {
        return radius*radius*3.14159;
    }

    //main method
    public static void main(String[ ] args)
    {
        //create a Circle object
        TestCircle myCircle = new TestCircle( );
        System.out.println("The area of the circle or radius " + myCircle.radius +
            " is " + myCircle.findArea( ) );
    }
}
```



In this revised program, **radius** and **findArea()** are members of the **TestCircle** class.

Since TestCircle contains a **main** method, it can be executed by Java.

The **main** method creates **myCircle** to be an instance of **TestCircle** and displays **radius** and **findsArea()** in **myCircle**.

You must always create an object before manipulating it.

Manipulating an object that has not been created causes a **NullPointerException** exception.



Constructors

One problem with the Circle class that we just discussed is that all of the objects created from it have the same radius of 1.0.

It would be more useful to create circles that could have a different radius.

Java enables us to define a special method in the class that will initialise an object's data.

This is known as a **constructor**.

You can use a **constructor** to assign an initial value to radius when creating an object

The **constructor** has exactly the same name as the class it comes from.

Constructors can be **overloaded**, making it easier to construct objects with different kinds of initial data values.

We will add the following constructors to the Circle class ... and then see what happens....

```
Circle(double r)
{
    radius = r;
}
```

```
Circle()
{
    radius = 1.0;
}
```


The Constructor Method

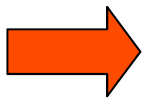
A constructor is a special method that initialises the instance variables of a class object.

A class's constructor method is called automatically when an object of that class is created, i.e., **instantiated**.

It is common to have **several constructors** for a class.

This is accomplished through **method overloading**.

Constructors can take arguments but cannot return an argument.



Attempting to declare a return type for a constructor and/or attempting to return a value from a constructor is a syntactic error.

When creating a new **Circle** object that has a radius of 5.0, we can simple assign 5.0 to `myCircle.radius` as follows:

```
myCircle = new Circle(5.0);
```

If we were to create a circle using the following statement in Java, the second constructor is used, which assigns the default radius to `myCircle.radius`:

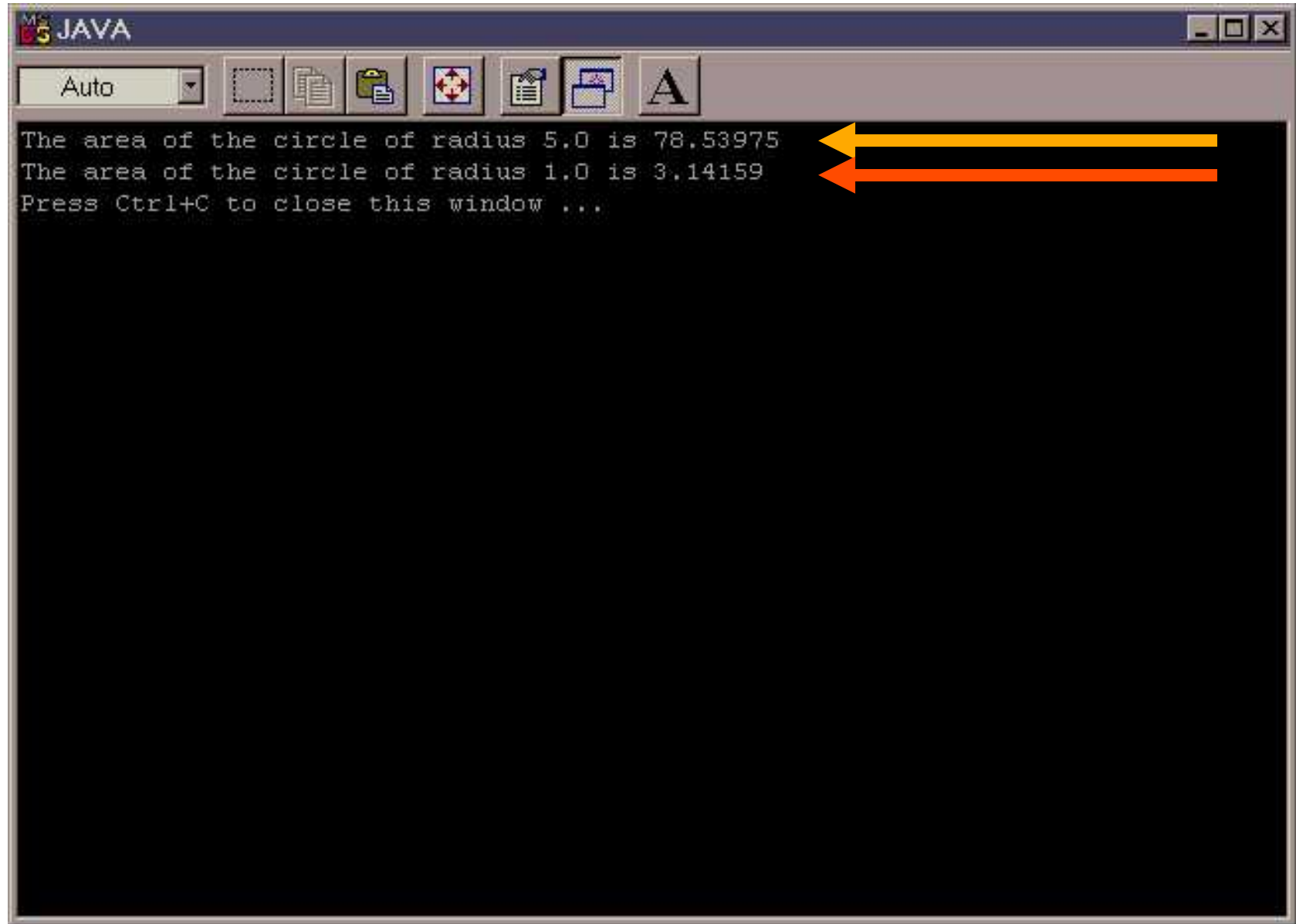
```
myCircle = new Circle( );
```

Now you know why the object is created with the syntax **ClassName()**.

This syntax is used to call a **constructor**.

If the class has no constructor, a default constructor (one that takes no arguments) is used, which will initialise the object's data to some default value specified in the default constructor

This means that if you do not use constructors all of your objects will be the same initially.



TextPad - [C:\TB\SoftDev-I_JAVA\myPrograms\TestCircleWithConstructors.jav

File Edit Search View Tools Macros Configure Window Help

1 class TestCircleWithConstructors
2 |
3 public static void main(String[] args)
4 {
5 //Test Circle with radius 5.0
6 Circle myCircle = new Circle(5.0);
7 System.out.println("The area of the circle of radius "
8 | myCircle.radius | " is " + myCircle.findArea());
9
10
11 //Test Circle with default radius
12 Circle yourCircle = new Circle();
13 System.out.println("The area of the circle of radius "
14 + yourCircle.radius + " is " + yourCircle.findArea());
15
16 //pause
17 System.out.println("Press Ctrl+C to close this window ...");
18 MyInput.readInt();
19 |
20 }
21
22 class Circle
23 {
24 double radius;
25
26 Circle(double r)
27 {
28 radius = r;
29 |
30 }
31
32 Circle()
33 {
34 radius = 1.0;
35 |
36 }
37
38 double findArea()
39 {
40 return radius*radius*3.14159;
41 |
42 }
43 }
44
45
46
47
48

Current Results
TestCircleWithC...

ASCII Characters

| | |
|----|----|
| 33 | ! |
| 34 | " |
| 35 | # |
| 36 | \$ |
| 37 | % |
| 38 | & |
| 39 | ' |
| 40 | (|
| 41 |) |
| 42 | * |
| 43 | + |
| 44 | , |
| 45 | - |
| 46 | . |
| 47 | / |
| 48 | 0 |

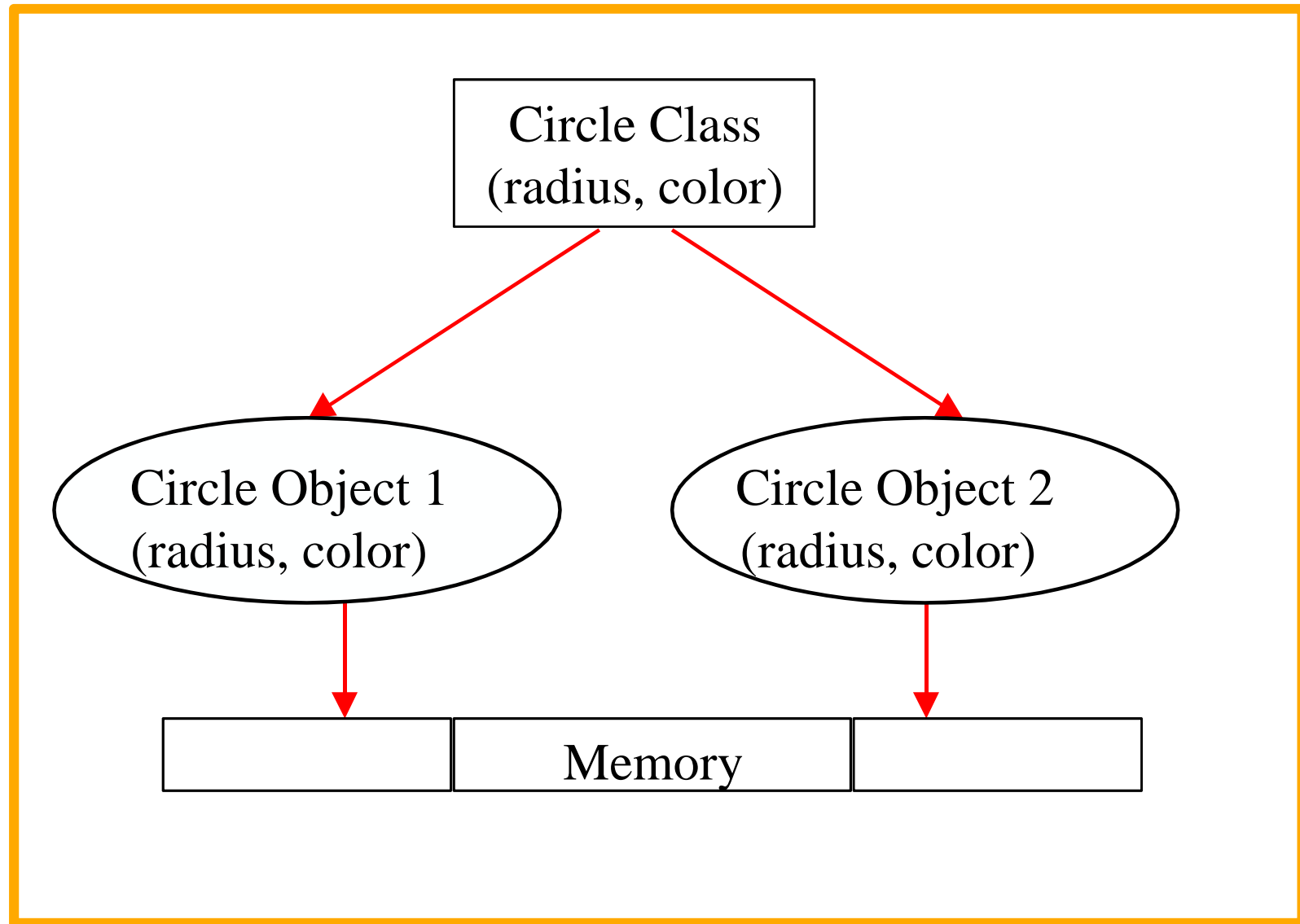
2/ 16 Home Lvt Block Sync Rec Caps 23:30

The new Circle class has two constructors.

You can specify a radius or use the default radius to create a Circle object.

The two objects (**myCircle** and **yourCircle**) have different data but share the same methods.

There we can compute their respective areas by using the **findArea()** method.



Object Oriented Programming (OOP) encapsulates data (attributes) and **methods** (behaviors) into **objects**.

The data and methods of an object are tied together.

Objects have the property of **information hiding**.



This means that although objects may **know how to communicate with one another** across well-defined interfaces, objects normally are **not allowed to know how other objects are implemented**.

Implementation details are hidden within objects themselves.

Consider a car as an object.

It is possible to drive a car without knowing the details of how the engine, gears, electronics work internally.

The concept of information hiding is important for good software design.

The class as the unit of programming in Java

In Java the unit of programming is the **class** from which all objects are eventually **instantiated**, that is, **created**.

Java programmers' concentrate on creating their own user-defined types called classes.

Classes are also referred to as **programmer defined types**.

Each **class** contains **data** as well as the **set of methods** that manipulate that data.

The **data** components of a class are called **instance variables**.

The **nouns** in a systems requirement document help the Java programmer determine an initial set of **objects** that will work together to implement the system.

Remember:

- All Java **objects** are **passed call by reference**. Only a memory address is passed, not a copy of the object.
- It is important to write programs that are **easy to maintain** and **easy to read** and **understand**. Code should be written with this in mind.

Modifiers

Java provides modifiers to control access to **data**, **methods** and **classes**.

static

Defines data and methods. It represents class-wide information that is shared by all instances of the class.

public

Defines classes, methods, and data in such a way that all programs can access them.

private

Defines methods and data in such a way that the declaring class can access them, but not any other class.

1

Get the **programs** from this lecture working

including ...

Circle, myCircle, yourCircle etc.