Student Name: _____________ __ __ __ **Roll No: ___________ __** **Section: __________ __**

Lab No. 01

Lab 01 – Introduction to Object Oriented

Lab Objectives:

1. Introducing Object Oriented
2. Object and Classes
3. Introduction to UML
4. Introduction to starUML
5. Creating Diagrams
6. Generating Code

1. Introducing Object Oriented

Everyone knows what an object is—a tangible thing that we can sense, feel, and manipulate. The earliest objects we interact with are typically baby toys. Wooden blocks, plastic shapes, and over-sized puzzle pieces are common first objects. Babies learn quickly that certain objects do certain things: bells ring, buttons press, and levers pull. The definition of an object in software development is not terribly different. Software objects are not typically tangible things that you can pick up, sense, or feel, but they are models of something that can do certain things and have certain things done to them. Formally, an object is a collection of data and associated behaviors. So, knowing what an object is, what does it mean to be object-oriented? Oriented simply means directed toward. So object-oriented means functionally directed towards modeling objects. This is one of the many techniques used for modeling complex systems by describing a collection of interacting objects via their data and behavior

If you've read any hype, you've probably come across the terms object-oriented analysis, object-oriented design, object-oriented analysis and design, and object oriented programming. These are all highly related concepts under the general object-oriented umbrella.

In fact, analysis, design, and programming are all stages of software development. Calling them object-oriented simply specifies what style of software development is being pursued.

Object-oriented analysis (OOA) is the process of looking at a problem, system, or task (that somebody wants to turn into an application) and identifying the objects and interactions between those objects. The analysis stage is all about what needs to be done. The output of the analysis

Student Name: _______________________ Roll No: ________________ Section: _____________

Stage is a set of requirements. If we were to complete the analysis stage in one step, we would have turned a task, such as, I need a website, into a set of requirements.

For example: Website visitors need to be able to (italic represents actions, bold represents objects):

• *review* our **history**

• *apply* for **jobs**

• *browse, compare,* and *order* **products**

In some ways, analysis is a misnomer. The baby we discussed earlier doesn't analyze the blocks and puzzle pieces. Rather, it will explore its environment, manipulate shapes, and see where they might fit. A better turn of phrase might be object-oriented exploration. In software development, the initial stages of analysis include interviewing customers, studying their processes, and eliminating possibilities. **Object-oriented design (OOD)** is the process of converting such requirements into an implementation specification. The designer must name the objects, define the behaviors, and formally specify which objects can activate specific behaviors on other objects. The design stage is all about how things should be done. The output of the design stage is an implementation specification. If we were to complete the design stage in a single step, we would have turned the requirements defined during object-oriented analysis into a set of classes and interfaces that could be implemented in (ideally) any object-oriented programming language.

Object-oriented programming (OOP) is the process of converting this perfectly defined design into a working program that does exactly what the CEO originally requested.

2. Objects and Classes

So, an object is a collection of data with associated behaviors. How do we differentiate between types of objects? Apples and oranges are both objects, but it is a common adage that they cannot be compared. Apples and oranges aren't modeled very often in computer programming, but let's pretend we're doing an inventory application for a fruit farm. To facilitate the example, we can assume that apples go in barrels and oranges go in baskets.

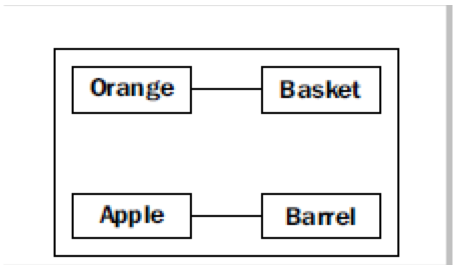
Now, we have four kinds of objects: apples, oranges, baskets, and barrels. In object-oriented modeling, the term used for *kind of object* is **class**. So, in technical terms, we now have four classes of objects.

What's the difference between an object and a class? Classes describe objects. They are like blueprints for creating an object. You might have three oranges sitting on the table in front of you.

Student Name: ______________________ Roll No: _____________ Section: __________

Each orange is a distinct object, but all three have the attributes and behaviors associated with one class: the general class of oranges.

The relationship between the four classes of objects in our inventory system can be described using a **Unified Modeling Language** (invariably referred to as **UML**, because three letter acronyms never go out of style) class diagram. Here is our first class diagram:



This diagram shows that an Orange is somehow associated with a Basket and that an Apple is also somehow associated with a Barrel. Association is the most basic way for two classes to be related

3. Introduction to UML

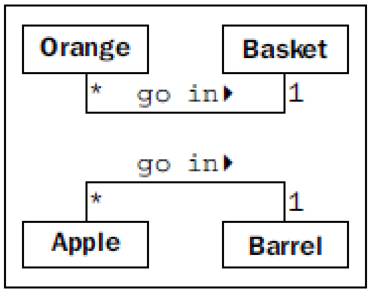
UML is very popular among managers, and occasionally disparaged by programmers. The syntax of a UML diagram is generally pretty obvious; you don't have to read a tutorial to (mostly) understand what is going on when you see one. UML is also fairly easy to draw, and quite intuitive. After all, many people, when describing classes and their relationships, will naturally draw boxes with lines between them. Having a standard based on these intuitive diagrams makes it easy for programmers to communicate with designers, managers, and each other.

Our initial diagram, while correct, does not remind us that apples go in barrels or how many barrels a single apple can go in. It only tells us that apples are somehow associated with barrels. The association between classes is often obvious and needs no further explanation, but we have the option to add further clarification as needed.

The beauty of UML is that most things are optional. We only need to specify as much information in a diagram as makes sense for the current situation. In a quick whiteboard session, we might just quickly draw lines between boxes. In a formal document, we might go into more detail. In the case of apples and barrels, we can be fairly confident that the association is, many apples go in one

Student Name: ______________________ Roll No: ______________ Section: __________

barrel, but just to make sure nobody confuses it with, one apple spoils one barrel, we can enhance the diagram as shown:



This diagram tells us that oranges go in baskets with a little arrow showing what goes in what. It also tells us the number of that object that can be used in the association on both sides of the relationship. One Basket can hold many (represented by a *) Orange objects. Any one Orange can go in exactly one Basket. This number is referred to as the multiplicity of the object. You may also hear it described as the cardinality. These are actually slightly distinct terms. Cardinality refers to the actual number of items in the set, whereas multiplicity specifies how small or how large this number could be.

4. Introduction to starUML

4.1 Introduction

In this short guide, we'll be looking at StarUML, which we'll be using for the purpose of drawing UML diagrams. We tell you how to start StarUML and how to create the types of diagram which you'll meet on the module. Notice that if you haven't come across some of these diagrams before, the terminology used might not seem to make sense—rest assured that it will once you've learnt about the diagrams themselves.

4.2 Starting StarUML

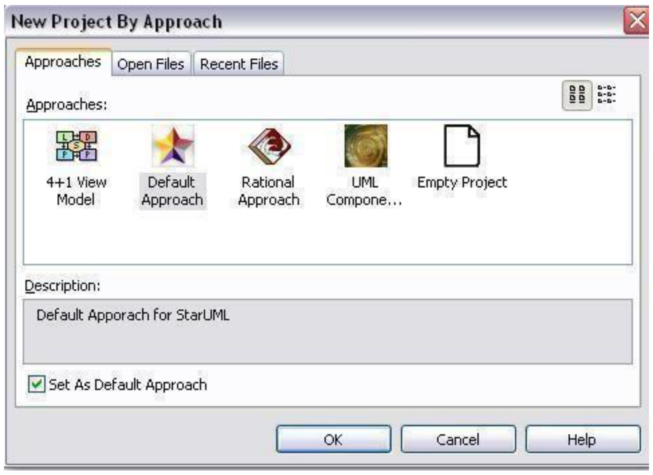
StarUML is a CASE tool. As such it lest you draw UML diagrams, but there is a lot more to it than that. For one thing, it keeps track of the "things" you create; even when something appears on more than one diagram, provided you use it correctly, as we shall see. It also lets you generate code from your UML diagrams in a number of languages, including Java.

StarUML is a native Windows application. This means that it runs more quickly than some other tools, but it also means that you can only run it under Windows. It does, however, run

Student Name: ______________ Roll No: ___________ Section: ________

quickly enough to be usable on a version of Windows running on a virtual machine such as Parallels Desktop or Virtual Box.

You start StarUML from the Windows Start Menu. When you do so you will see the *New Project* dialog box (figure 1). It offers a number of "approaches"—these affect the way in which the diagrams are managed in the Model Explorer part of the main window.



Default Apporach for StarUML

☑ Set As Default Approach

Figure 1: New Project Dialog Box

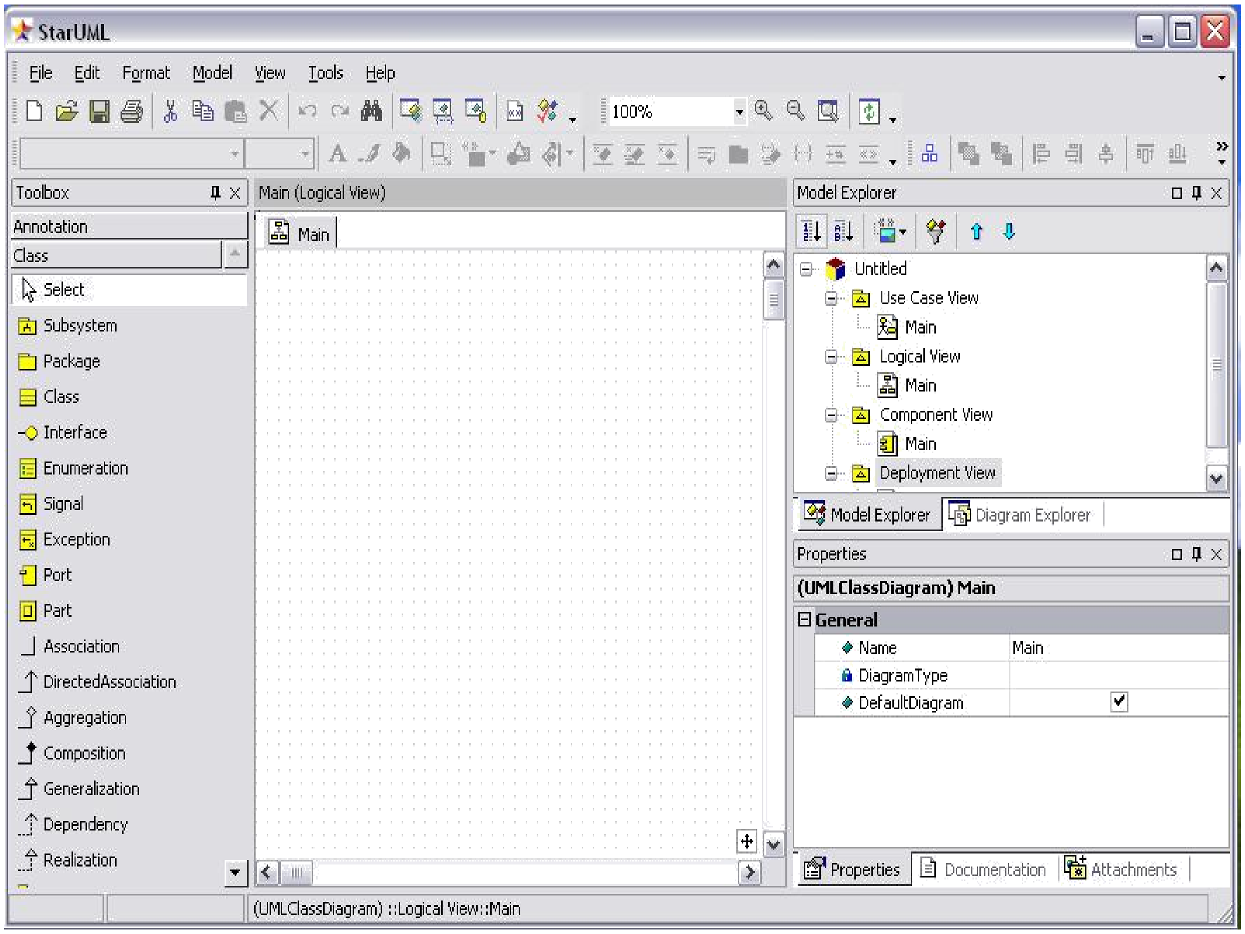
For now, select the *Rational Approach*, and click OK. You will now see the main window of Star UML, as seen in figure 2. Notice the contents of this window—you will see the *Toolbox* at the left, the *Model Explorer* and *Properties* windows at the right, and in the middle, the drawing window where you create the diagrams. Initially there is a tab marked "Main" which represents a class diagram in the Logical View.

Student Name: ______________ Roll No: __________ Section: ________

5. Creating Diagrams

5.1 Class Diagrams

We'll start by looking at the *Model Explorer* —see figure 3. The unexpanded view is shown on the left; if we expand each of the nodes the view changes to that shown on the right.



Properties

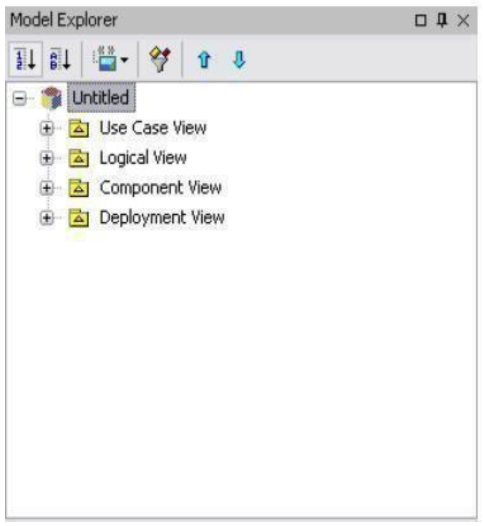
(UMLClassDiagram) Main

General

Name	Main
DiagramType	
DefaultDiagram	✓

(UMLClassDiagram) ::Logical View::Main

Student Name: ______________________ Roll No: ______________ Section: ____________



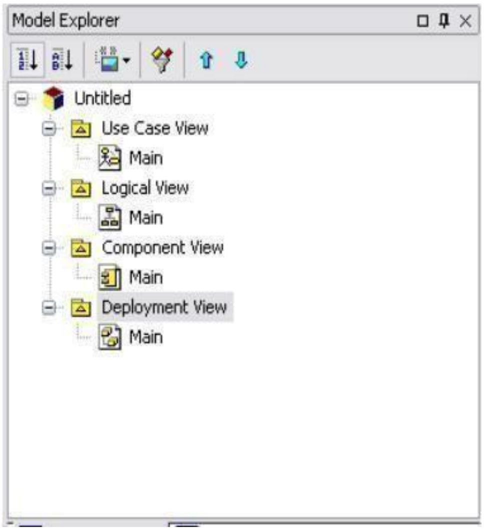


Figure 3: Model Explorer

To populate the class diagram, we click on one of the tools in the toolbox—see figure 4. Then click at the place within the diagram area where we want the object to be created. An instance of the type of object we selected will then appear at that point on the diagram—obviously you will want to change the default name (unusually, the names appear to be those of StarUML's creators, used in turn).

We can create further elements by repeating this process. In order to link elements, we simply click on one of the appropriate symbols in the toolbox, then, on the diagram, drag from the source of the association (e.g.) to the target. We can then select the line and view its properties in the *Properties Window* at the bottom right of the screen; here we can name the association, set the multiplicities and so on.

Adding attributes and operations to a class is also straightforward. We right-click on the class concerned, then select *Add…* and *Attribute*. We can then type the name of the attribute to replace the offered default. Notice that we can precede the name by a visibility symbol, and follow it with a colon and the name of the attribute's type, and these will be recognized as such (we'll see what is meant by these and how the visibility and type are depicted on a UML class diagram in due course).

Student Name: ______________ **Roll No:** __________ **Section:** __________

Notice that as we create classes, they are shown in the Model Explorer. If we want to add the same "thing" (I don't want to use the word "object", as it has a specific meaning in UML!) to more than one

diagram, we can drag it on to the diagram from the Model Explorer. We'll see an example of this when we look at creating Sequence Diagrams. See figure 5. If we add something by mistake, we must select it in Model Explorer and then select Delete *from model* from Model Explorer. If we just delete it from the diagram, it stays in the model itself—beware!

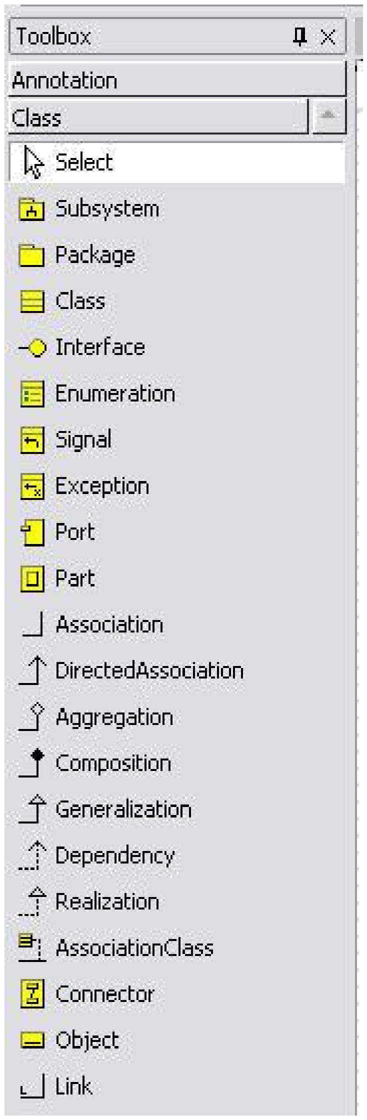


Figure 4: Toolbox

Student Name: _____________ _ _ __ Roll No: ___________ _ Section: __________ _

5.2 Use Case Diagrams

You will find it in next year—but obviously now as you have use StarUML to create such a diagram. As it happens the process is quite straightforward, and not dissimilar from creating a class diagram—although obviously the contents of the Toolbox will be different. In fact, Start UML has created an empty use case diagram called "Main" for you—logically enough, it's within the Use Case View in Model Explorer.

Double-click on the diagram's icon within *Model Explorer* to show the diagram. Then click on the "System Boundary" element, and draw it at an appropriate size on the diagram by dragging the mouse— the outline of the element is shown as a dashed rectangle. You can then click on other types of elements and add them.

You can use an "Association" element to link an Actor with a use case; to relate use cases you can employ the "Include" and "Extend" links in the toolbox. Remember that you need to draw *from* the extending use case *to* the "parent" use case for an extends relationship, and *from* the parent *to* the included use case for an includes relationship.

5.4 Combine Fragments

These are the way in which selection and iteration are added to sequence diagrams. You add a Combined Fragment using the appropriate element in the toolbox. You will almost certainly need to change the "interaction operator"—the default is **seq**. Just select an alternative from the drop-down list.

You now need to set the guard for what StarUML calls the "interaction operand". You do **NOT** use the "Interaction Operand" element in the toolbox for this (but see below) as one has already been created— you just need to select it. Some-times clicking on the Combined Fragment in the diagram achieves this, but the easiest way is simply to expand the Combined Fragment's node in *Model Explorer*, and select the Interaction Operand there. You can then type the guard into the appropriate text box in Properties.

So what *is* the Interaction Operand element in the toolbox used for, then? It's to add *additional* interaction operands to fragments which require more than one—in other words, **alt** fragments. Figure 6 shows an example of a simple sequence diagram created with StarUML.

6. Generating Code

Wouldn't it be nice if a tool would do *more* than let us draw diagrams (and, perhaps, make sure they are correct). Well. many CASE tools will generate code from us, and StarUML is no exception. Just don't expect too much—all that can really be generated are the outlines of classes—in other words the classes themselves and their methods, as seen on the class diagram.

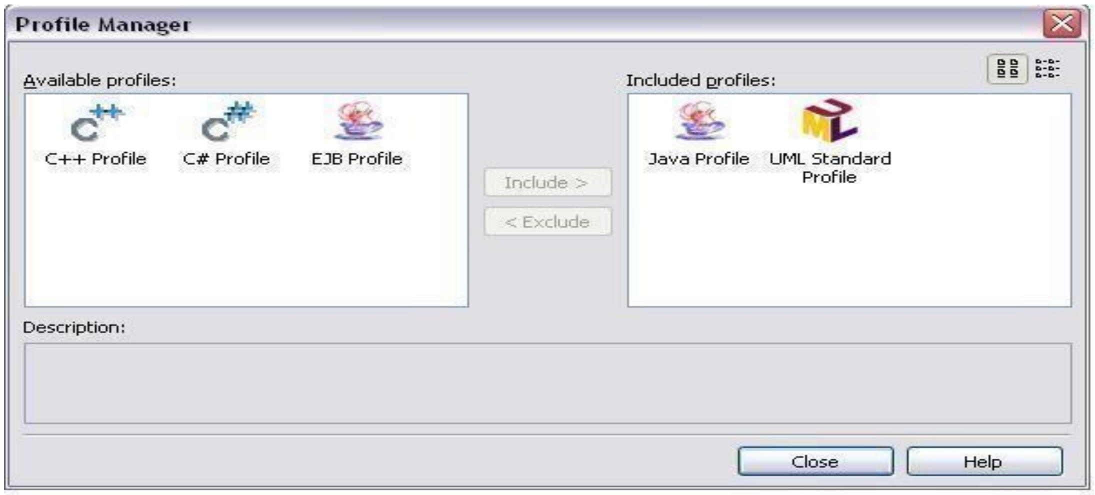


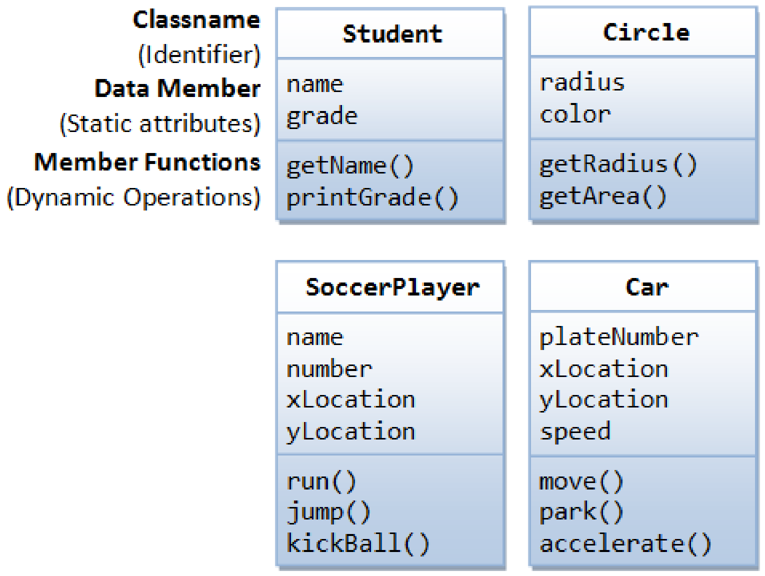
Figure 7: Profile Manager

Student Name: ______________ **Roll No:** __________ **Section:** __________

Programming Exercise (OOAD)

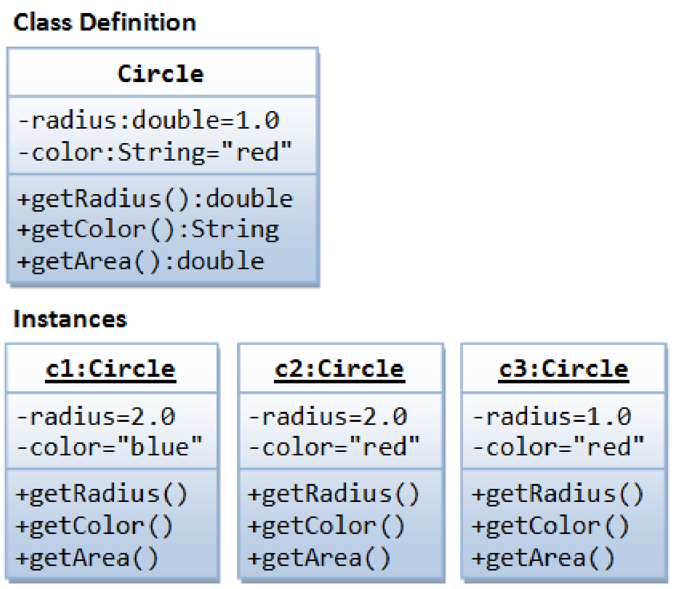
1. Draw simple class diagrams of the following classes:
   a. Class Student
   b. Class Circle
   c. Class Soccer Player
   d. Class Car

Once you have created the class add the attributes and methods(), accordingly. For your help the following diagram is given.



2. Suppose you need to create many circles of different colors and radius for this you need to create a class name it as Circle. After creating class add two attributes such as radius and color. You can use access modifiers to your class attributes such as private (-), public (+) and protected (#). You can add two methods to the class Circle that can be getting the value of radius from the user and the color. So now create two methods named it as getRadius() and getColor(). Once we can create the class we are able to make as many as objects as possible which are known as instances. For your understanding following is the figure which elaborate the idea of class Circle with three instances and each have different values.

Student Name: ______________ **Roll No:** ____________ **Section:** __________


Class Definition
Circle
-radius:double=1.0
-color:String="red"
+getRadius():double
+getColor():String
+getArea():double
Instances
c1:Circle
-radius=2.0
-color="blue"
+getRadius()
+getColor()
+getArea()
c2:Circle
-radius=2.0
-color="red"
+getRadius()
+getColor()
+getArea()
c3:Circle
-radius=1.0
-color="red"
+getRadius()
+getColor()
+getArea()