Lab Objective

1. **Familiarize** students with the implementation of classes 2.
2. **Instantiation** of objects accordingly. **Lab Activities**

A. Writing Class Definition

- Define a class 'Icecream' that has the following instance variables:
  a. icecreamType (String)
  b. icecreamCompany (String)
  c. icecreamPrice (double)
- At the beginning, the access specifier is default (no public or private while declaring them).
- Define appropriate constructors for 'Icecream' class that set the attributes values. One constructor should take attributes value as parameter.
- Define the following instance methods:
  a. toString(): returns a string containing an icecream information.
  .

B. Demonstrating Class Functionalities

- Define the Main (Driver) class that has the main() method.
- Create two icecream objects using both constructors.
- Print their values.

C. Introducing private access specifier and Setters (Mutators) and Getters (Accessors)

- While declaring instance variables, using private access modifier indicates that the instance variable cannot be accessed from outside the class definition. It is helping to achieve data hiding that states that the data of an object must not be accidentally modified or updated.
- Therefore, to access the private instance variables from outside the class, we need to include a few more instance methods known as setters and getters.
- **Setters are used to set the value.**

```
void setIcecreamType(String icecreamType){
icecreamType = icecreamType;
}
```

- **Getters are used to return the value.**

```
String getIcecreamType(){
 Return icecreamType;
}
```

- **Modify your Icecream class definition accordingly by including setters and getters.**
- Remove constructors for 'Icecream' class in previous example and define new constructor that set price of the Icecream class to zero.

D. Passing Objects to Methods

- In Java we can pass objects to methods
- When we pass a primitive type to a method, it is passed by value.
- But when we pass an object to a method, the situation changes dramatically, because objects are passed by what is effectively call-by-reference.
- **Add equals (Icecream I) method to your Icecream class and return true if price of caller object and callee object are same; false, otherwise.**
- **Add compareTo(Icecream I): returns 1 if price of caller object is higher, 0 if both prices are the same, 1 otherwise**

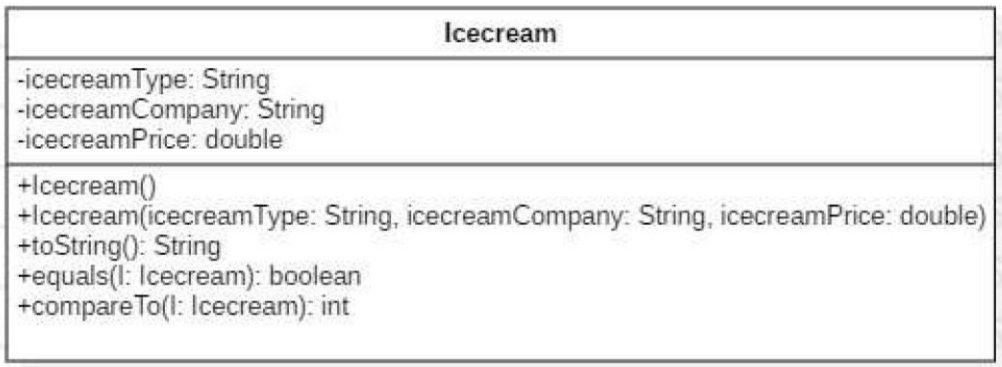
E. Array of Objects

- An array of objects is actually an array of reference variables.
- For the Icecream class we can create array of objects:

Icecream[] IcecreamArray = new Icecream [10];

F. Introducing Class Diagram

- Specifications of a class can be also expressed using a class diagram.
- Class diagram is a standard way to represent a class.
- The above-mentioned Icecream class can be represented using the following diagram.



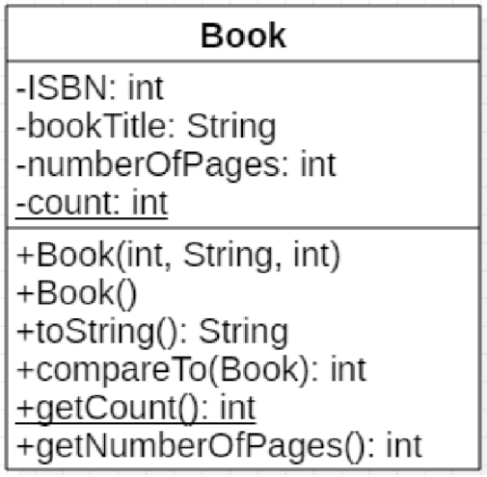
Lab Problems

01: Based on the 'Icecream' class as defined before, now create an array of Icecream type objects and get the input from the user to initialize those objects. The size of array must be at least 5.

Now, write a static method searchByCompany in the Main class of the previous implementation, which takes a String parameter representing company name and then prints all icecream's information manufactured by that company.

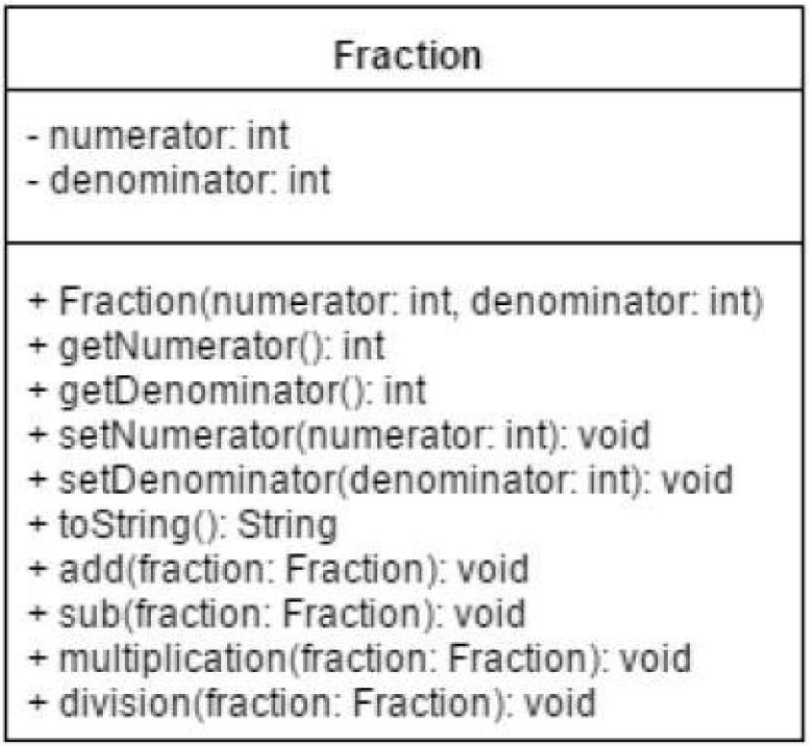
02: Write a program in Java that follows the specifications as given below.

- Define a class **Book** as shown in the following figure.



- Create your *Main* class and the *main()* method. Define an array of Book type objects of size 5.
- Instantiate these Book objects by taking the inputs from the user.
- Print all Book objects' data using a static method *displayAll()*.
- Invoke *compareTo()* method to compare any two Book objects based on their pages. If caller object's number of pages is greater than callee object's number of pages, compareTo() returns 1; if both pages are same, the method returns 0 and -1 otherwise. Make sure to print the returned value.
- Define a static method *isHeavier()* within the *Main* class that takes a Book object as input parameter and returns true if the Book's number of pages is greater than 500; false, otherwise. Add appropriate code into your main() method to demonstrate its functionalities.

03: Implement the following class 'Fraction' and test its methods.



void add(Fraction fraction)
Adds two Fraction objects and **stores the result** into **calling object**. This is how addition is performed for fractions: 1 / 4 + 3 / 5 = 1 * 5 + 3 * 4 / 4 * 5 = 17 / 20

String toString()
Returns the value of the fraction in 1 / 2 format where 1 is numerator and 2 is denominator.

Now write a test program, take two Fraction objects. Print both of them. Test add, sub, multiplication and division methods. Print calling object after each method call.