# Introduction to Java for C++ Programmers

Segment - 4

JAC 444

Professor: Mahboob Ali

# static key word

- **static keyword** in java used for memory management purposes, use it with methods, variables, blocks, nested classes.
  - Java static variables gets memory only once i.e. when class is loaded.
  - Java static property is shared in all objects.

```
class Student{
          int id;
          String name;
          String college = "ICT";
}
```

```
class Student{
      int id;
      String name;
      static String college = "ICT";
}
```

# this keyword

- The this keyword is the name of a reference that refers to an object itself.
- Common uses of this:
    1. One common use of the this keyword is reference a class's *hidden data fields*.
    2. Second is to enable a constructor to invoke another constructor of the same class.
    3. Third is to invoke a method of the current class.
    4. And others…..

```
private double radius;

public void setRadius(double radius){
     this.radius = radius;
}
```

```
private double radius = 1;

public void setRadius(double radius){
     radius = radius;
}
```

```
public class F {
  private int i = 5;
  private static double k = 0;

  public void setI(int i) {
    this.i = i;
  }

  public static void setK(double k) {
    F.k = k;
  }

  // Other methods omitted
}
```

```
Suppose that f1 and f2 are two objects of F.

Invoking f1.setI(10) is to execute
   this.i = 10, where this refers f1

Invoking f2.setI(45) is to execute
   this.i = 45, where this refers f2

Invoking F.setK(33) is to execute
   F.k = 33. setK is a static method
```

Invoking an Instance method

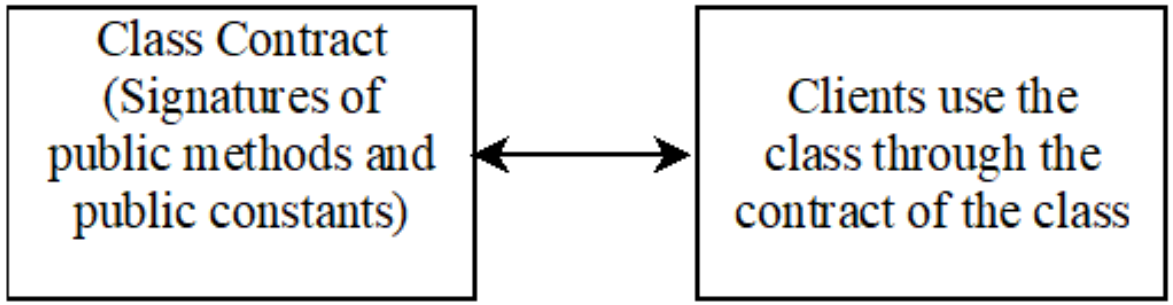
```
public class Circle{
     private double radius;

     public Circle(double radius){
          this.radius = radius; }

     public Circle(){
          this(1.0);}
     ……
```

The **this** keyword is used to reference the hidden data field radius of the object being constructed.

The **this** keyword is used to invoke another constructor.

# Abstraction

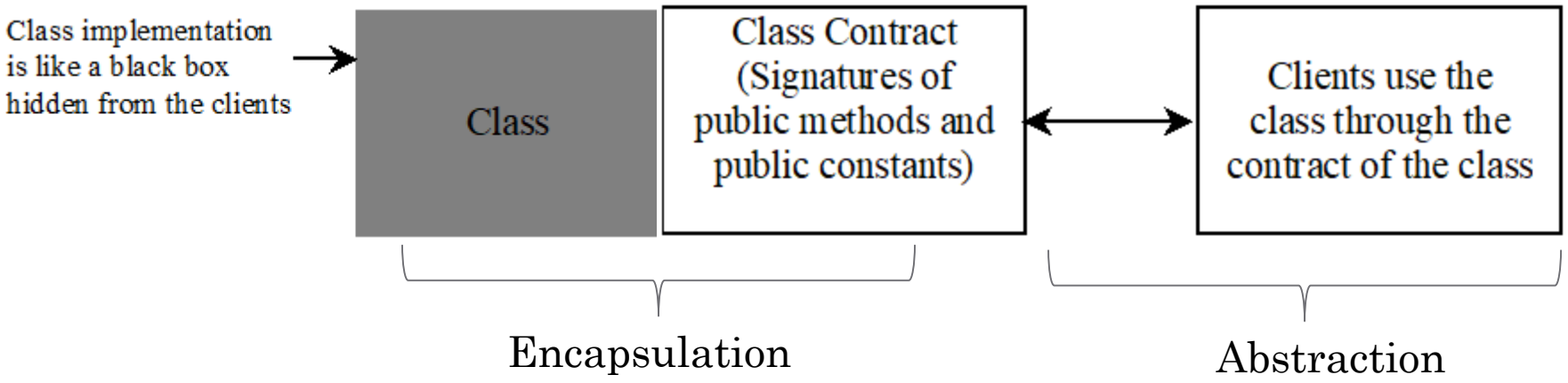
- Class abstraction means to separate class implementation from the use of the class.




- Class abstraction lets you focus on what object does instead of how it does it.

# Encapsulation

- The details of implementation are encapsulated and hidden from the user.




- Abstraction and encapsulation are two sides of the coins.

# Loan Class

Serves as the contract for the Loan class

| Loan | |
|---|---|
| -annualInterestRate: double | The annual interest rate of the loan (default: 2.5). |
| -numberOfYears: int | The number of years for the loan (default: 1) |
| -loanAmount: double | The loan amount (default: 1000). |
| -loanDate: Date | The date this loan was created. |
| +Loan() | Constructs a default Loan object. |
| +Loan(annualInterestRate: double, numberOfYears: int, loanAmount: double) | Constructs a loan with specified interest rate, years, and loan amount. |
| +getAnnualInterestRate(): double | Returns the annual interest rate of this loan. |
| +getNumberOfYears(): int | Returns the number of the years of this loan. |
| +getLoanAmount(): double | Returns the amount of this loan. |
| +getLoanDate(): Date | Returns the date of the creation of this loan. |
| +setAnnualInterestRate( annualInterestRate: double): void | Sets a new annual interest rate to this loan. |
| +setNumberOfYears( numberOfYears: int): void | Sets a new number of years to this loan. |
| +setLoanAmount( loanAmount: double): void | Sets a new amount to this loan. |
| +getMonthlyPayment(): double | Returns the monthly payment of this loan. |
| +getTotalPayment(): double | Returns the total payment of this loan. |

```
public class Loan {
  private double annualInterestRate;
  private int numberOfYears;
  private double loanAmount;
  private java.util.Date loanDate;

  /** Default constructor */
  public Loan() {
    this(2.5, 1, 1000);
  }

  /** Construct a loan with specified annual interest rate,
      number of years and loan amount
    */
  public Loan(double annualInterestRate, int numberOfYears,
      double loanAmount) {
    this.annualInterestRate = annualInterestRate;
    this.numberOfYears = numberOfYears;
    this.loanAmount = loanAmount;
    loanDate = new java.util.Date();
  }

  /** Return annualInterestRate */
  public double getAnnualInterestRate() {
    return annualInterestRate;
  }

  /** Set a new annualInterestRate */
  public void setAnnualInterestRate(double annualInterestRate) {
    this.annualInterestRate = annualInterestRate;
  }

  /** Return numberOfYears */
  public int getNumberOfYears() {
    return numberOfYears;
  }

  /** Set a new numberOfYears */
  public void setNumberOfYears(int numberOfYears) {
    this.numberOfYears = numberOfYears;
  }

  /** Return loanAmount */
  public double getLoanAmount() {
    return loanAmount;
  }

  /** Set a newloanAmount */
  public void setLoanAmount(double loanAmount) {
    this.loanAmount = loanAmount;
  }

  /** Find monthly payment */
  public double getMonthlyPayment() {
    double monthlyInterestRate = annualInterestRate / 1200;
    double monthlyPayment = loanAmount * monthlyInterestRate / (1 -
      (Math.pow(1 / (1 + monthlyInterestRate), numberOfYears * 12)));
    return monthlyPayment;
  }

  /** Find total payment */
  public double getTotalPayment() {
    double totalPayment = getMonthlyPayment() * numberOfYears * 12;
    return totalPayment;
  }

  /** Return loan date */
  public java.util.Date getLoanDate() {
    return loanDate;
  }
}
```

# Designing a Class

- (Coherence) A class should describe a single entity, and all the class operations should logically fit together to support a coherent purpose. You can use a class for students, for example, but you should not combine students and staff in the same class, because students and staff have different entities.

# Designing a Class, cont.

- (Separating responsibilities) A single entity with too many responsibilities can be broken into several classes to separate responsibilities.
- Classes are designed for reuse. Users can incorporate classes in many different combinations, orders, and environments.
- Provide a public no-arg constructor and override the equals method and the toString method defined in the Object class whenever possible.