



## INTRODUCTION TO JAVA

# Setup Environment

## ■ Jdk Download from

- <http://www.oracle.com/technetwork/java/javaee/downloads/java-ee-sdk-6u3-jdk-6u29-downloads-523388.html>

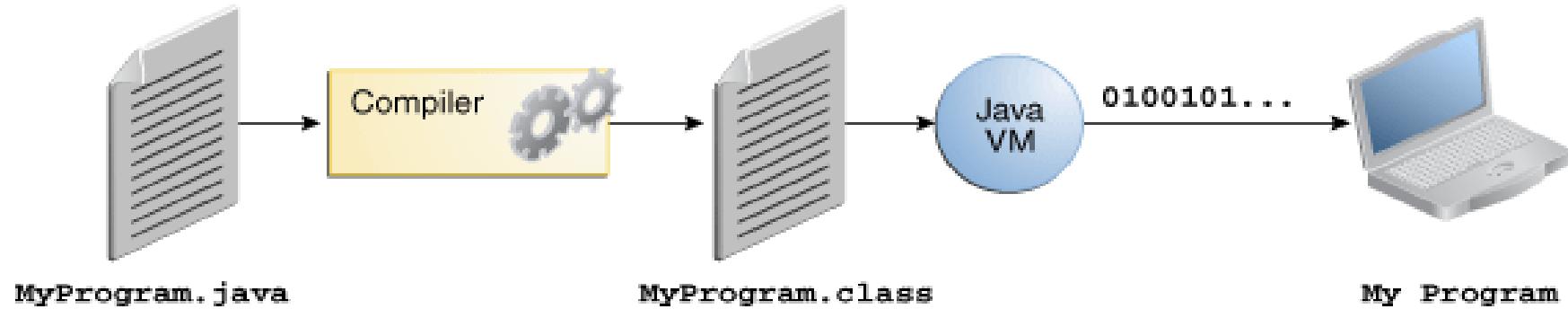
## ■ Eclipse Can be downloaded from

- <http://www.eclipse.org/downloads/>

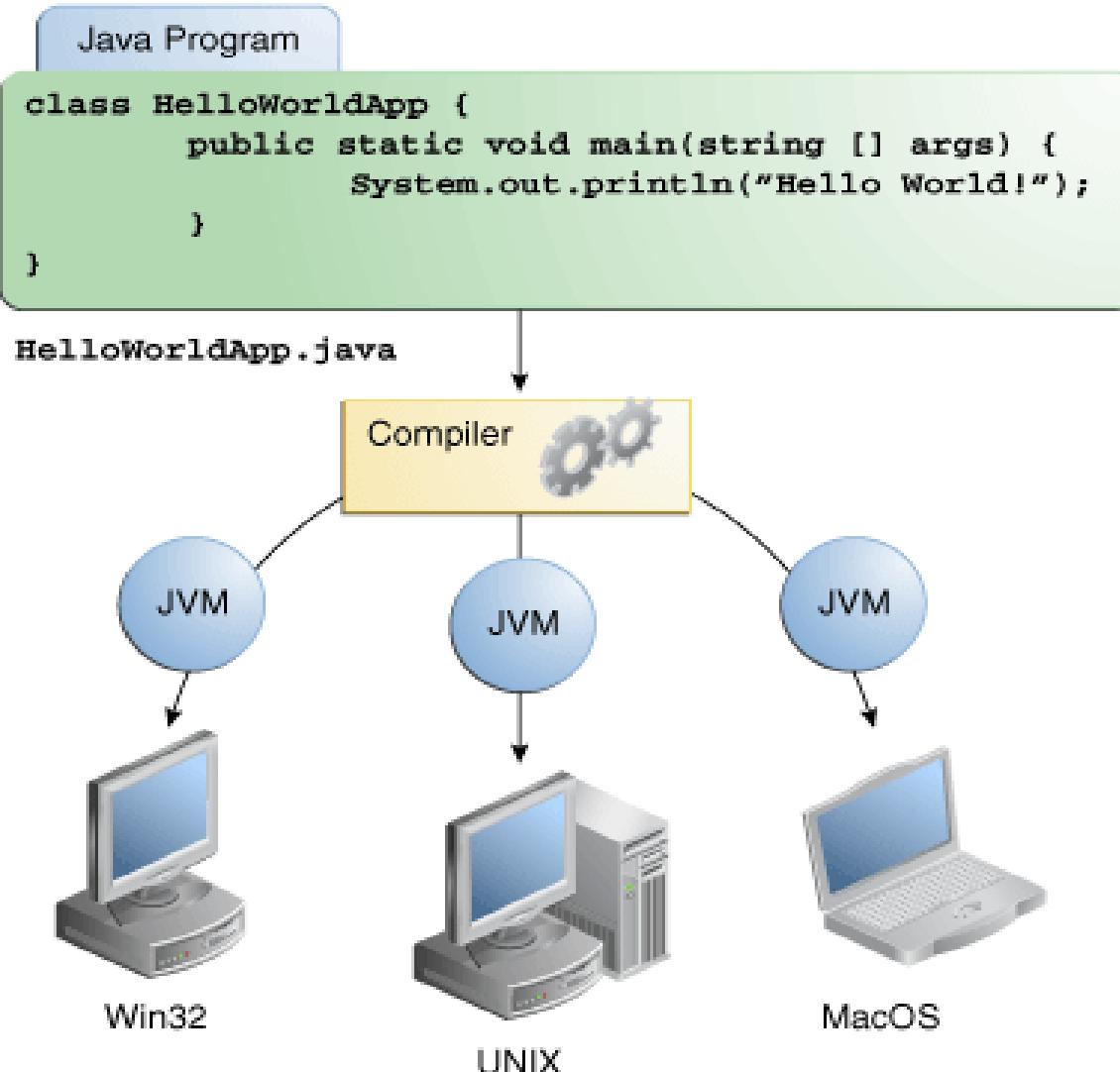
# Features

- Object Oriented
- Portable
- Multi Threaded
- Secure
- Platform independent

# Development Process



# Java Program

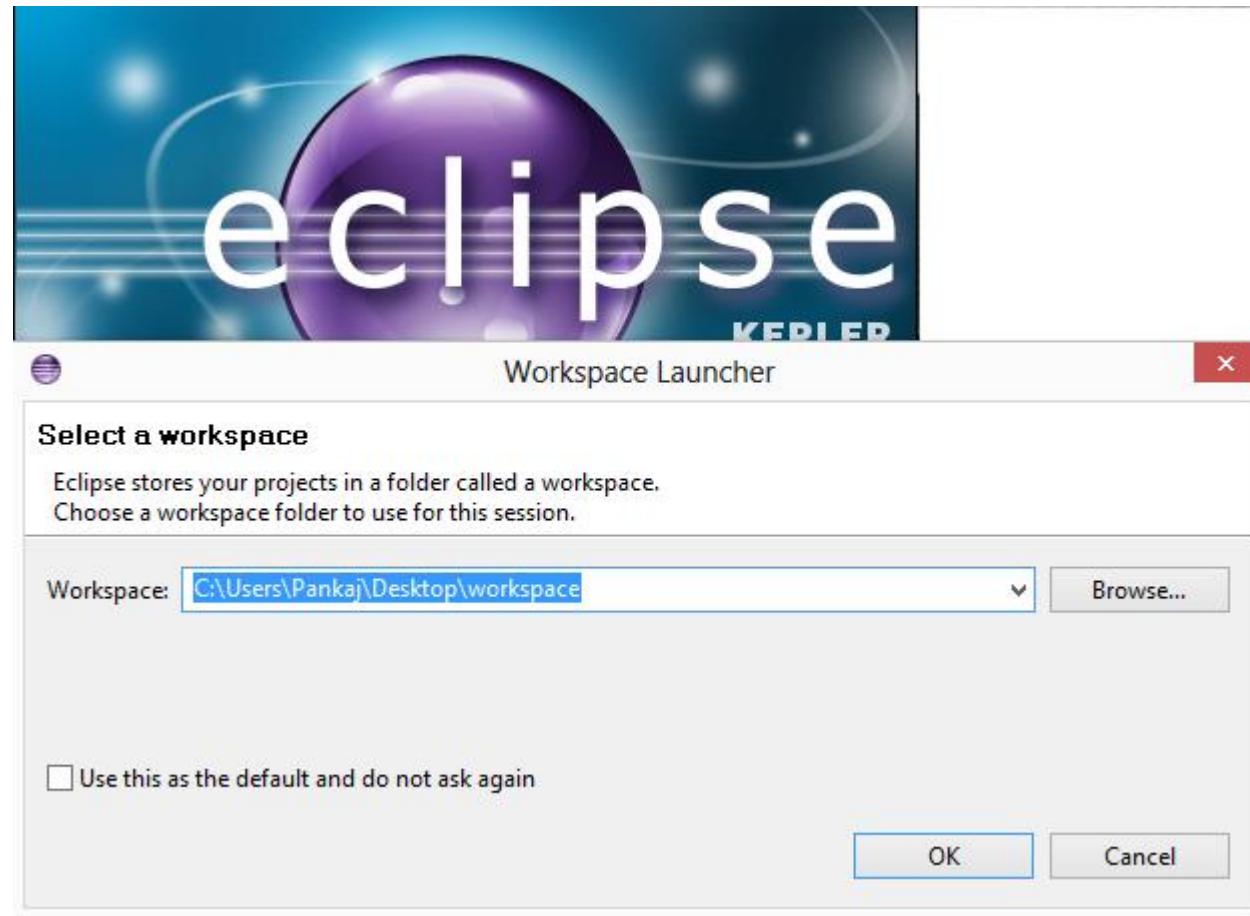


# Phases of Java Program

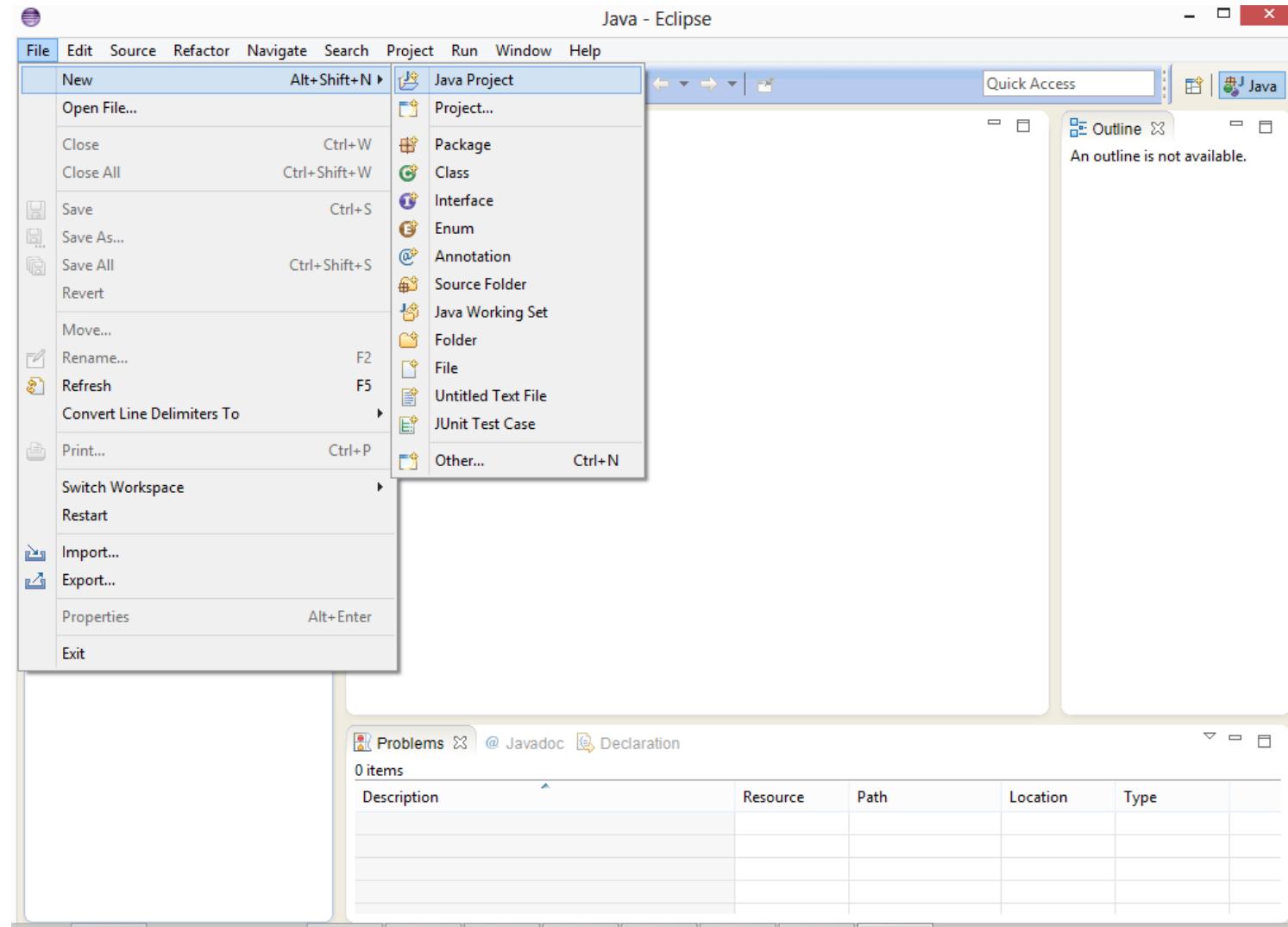
Task	Tool to use	Output
Write a program	text editor/eclipse	.java file
Compile the program	Java compiler	.class file bytecode
Run the program	Java Interpreter	Program Output

## Starting ‘Eclipse’

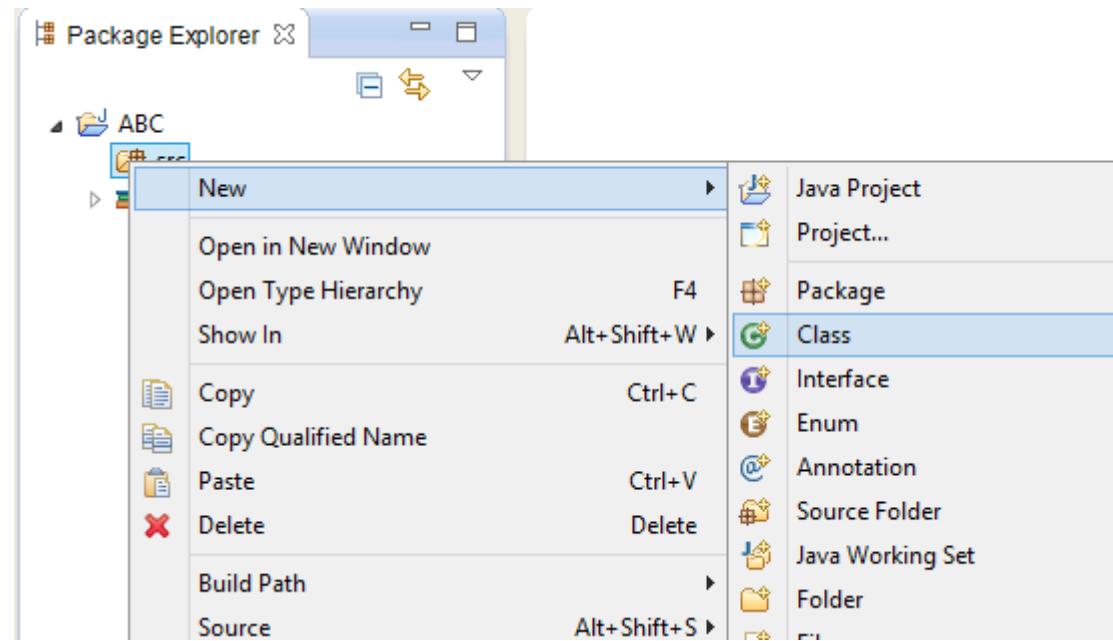
- 1) Click on Eclipse icon and specify your path name and then click on ‘workbench’.



## 2) Create 'new project'

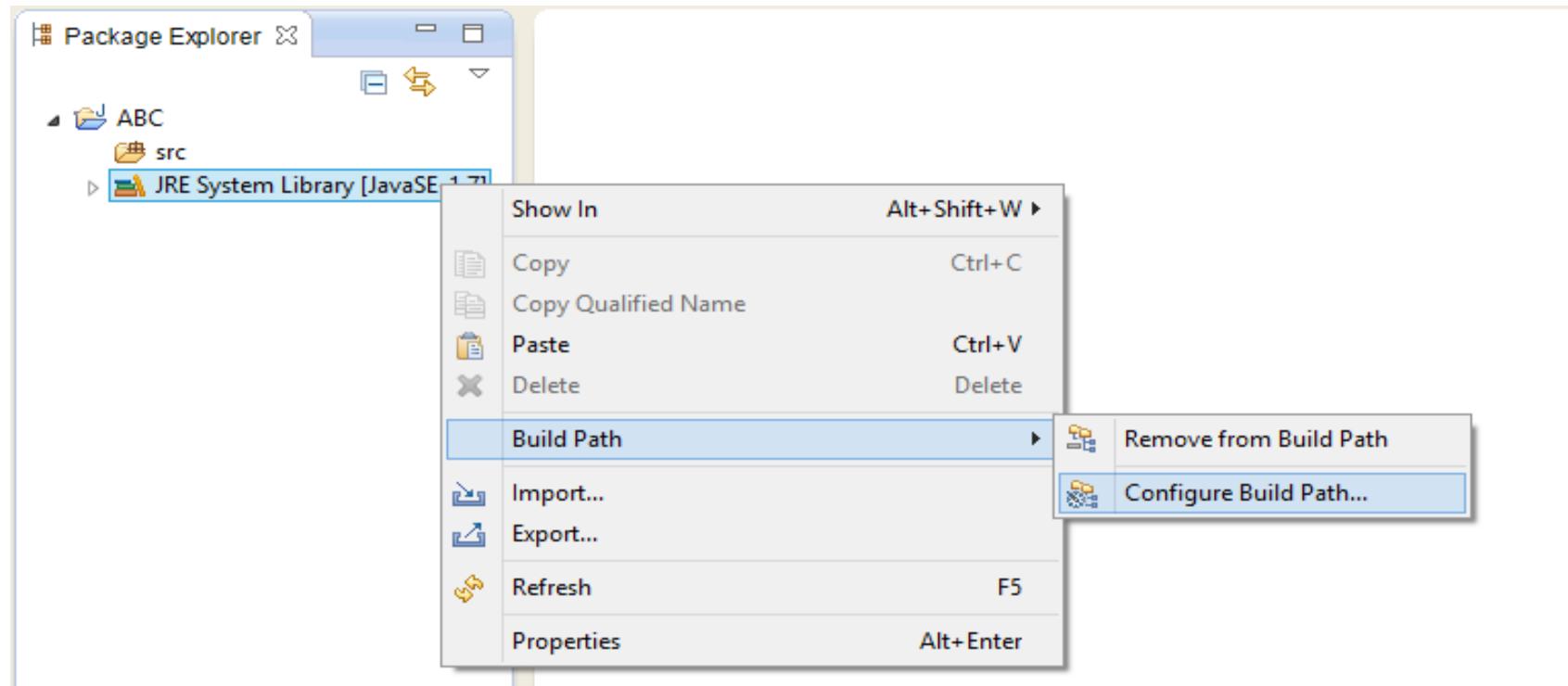


3) Create 'class' and write your java code in it.



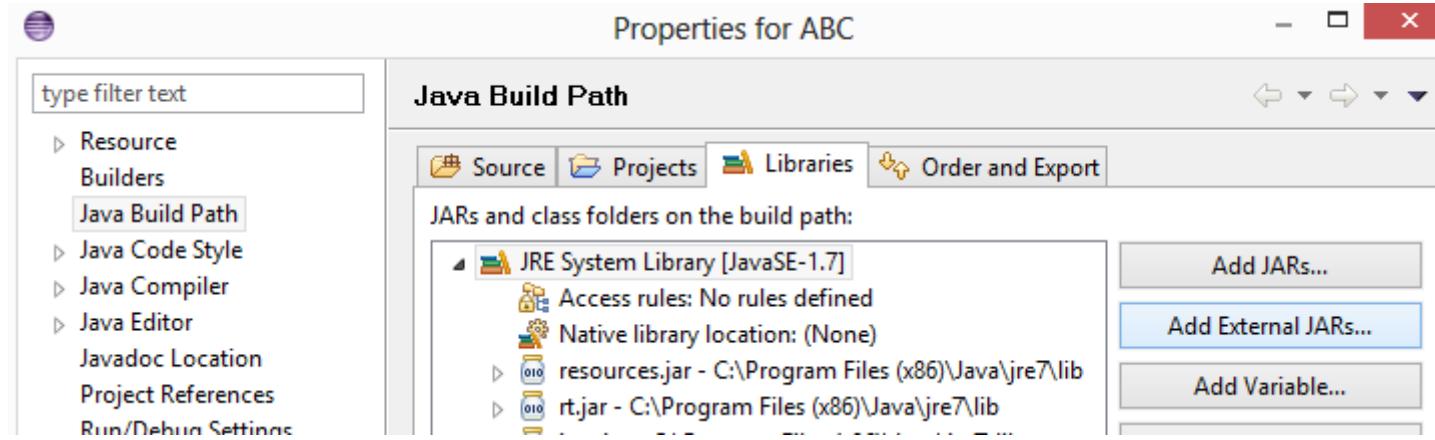
# Configuring Build Path

Sometimes while writing java codes for hadoop, you will need to import certain jar files into your build path.



## Build Path cont.

Select external jars to be added into your build path and click ok.



## Exercise 1

- After setting up jdk and eclipse, create a program to print your name, to ensure that both are installed properly

## Define Class

- Every module in java is class
- <modifier> class <name> {  
    <attributeDeclaration>  
    <constructor Declaration>  
    <method Declaration>

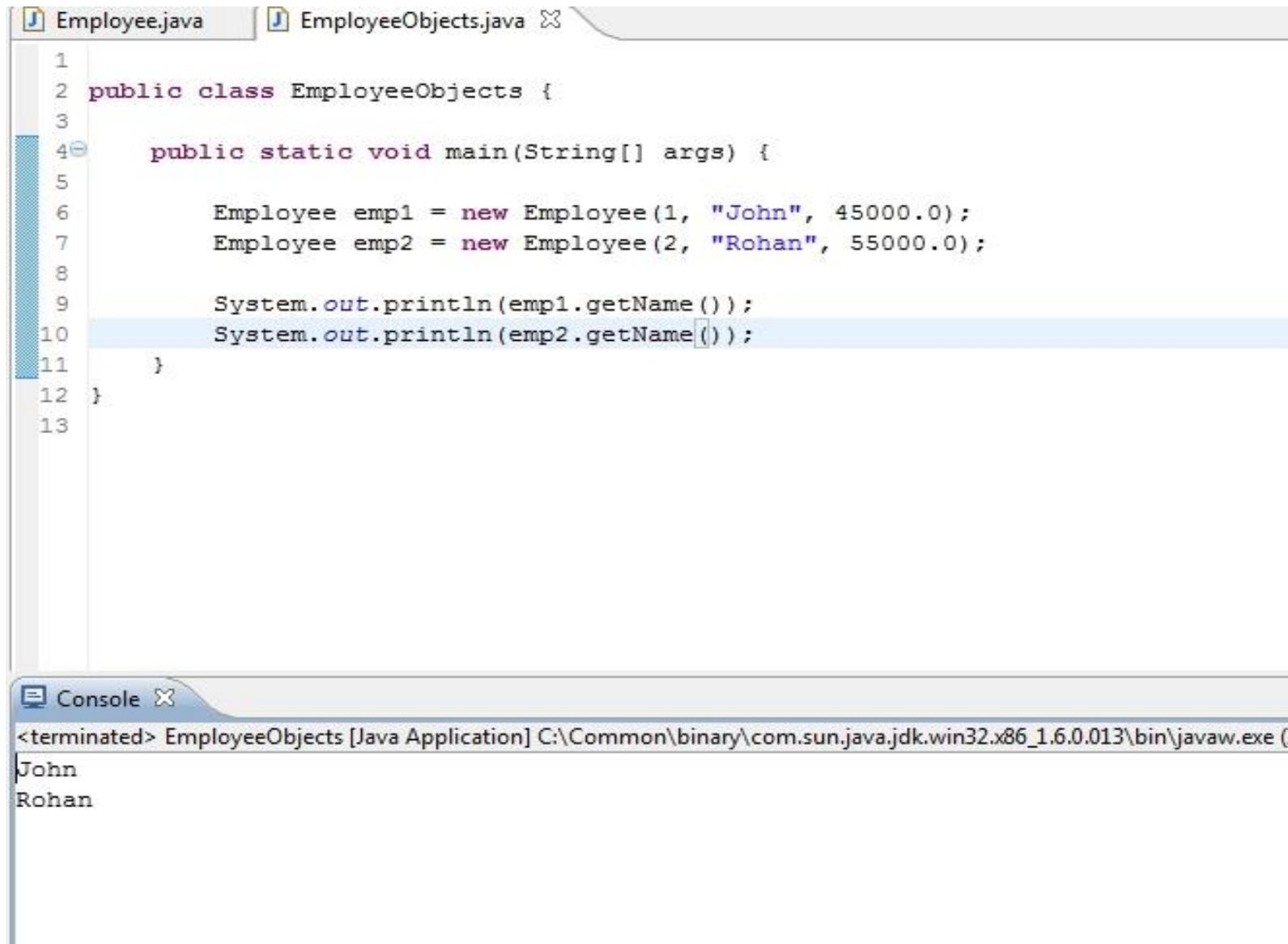
## Example

```
J Employee.java X
1
2 public class Employee {
3
4     private int id;
5     private String name;|
6     private double salary;
7
8     public Employee()
9     {
10    }
11
12     public int getId()
13     {
14         return id;
15     }
16
17     public String getName()
18     {
19         return name;
20     }
21
22     public double getSalary()
23     {
24         return salary;
25     }
26
27
28
29 }
```

# Instance Variable static variables

- Instance variables
  - Belong to object instance
  - Value of variable of an object instance is different from the values of other objects of same class
- Static variables
  - Variables that are associated with class
  - Same value for all object instances in same class

# Instance Variable



The screenshot shows a Java development environment with two files open:

- Employee.java**: A simple class definition.
- EmployeeObjects.java**: The main application code.

The **EmployeeObjects.java** code is as follows:

```
1 public class EmployeeObjects {
2
3     public static void main(String[] args) {
4
5         Employee emp1 = new Employee(1, "John", 45000.0);
6         Employee emp2 = new Employee(2, "Rohan", 55000.0);
7
8         System.out.println(emp1.getName());
9         System.out.println(emp2.getName());
10    }
11 }
12
13
```

In the **Console** tab, the output of the application is displayed:

```
<terminated> EmployeeObjects [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (3)
John
Rohan
```

## Static Variable

Employee.java EmployeeObjects.java

```
1
2 public class Employee {
3
4     private int id;
5     private String name;
6     private double salary;
7     private static int YEAR = 2014;
8     public Employee()
9     {
10    }
11
12     public Employee(int id, String name, double salary)
13    {
14         this.id = id;
15         this.name = name;
16         this.salary = salary;
17     }
18
19     public int getId()
20     {
21         return id;
22     }
23 }
```

# Methods

- Declare methods

- <modifier> <return type> <name> (<parameter>\*)

```
{  
    <statement>*  
}
```

## Getter Method

```
18  
19    public int getId()  
20    {  
21        return id;  
22    }  
23  
24    public String getName()  
25    {  
26        return name;  
27    }  
28  
29    public double getSalary()  
30    {  
31        return salary;  
32    }  
33
```

## Setter Method

- Mutator method

- Use to set or change value of variable
- Set<NameOfInstanceVariable>

```
public void setId(int id)
{
    this.id = id;
}

public void setName(String name)
{
    this.name = name;
}

public void setSalary(double salary)
{
    this.salary = salary;
}

public int getId()
{
    return id;
}
```

## Static Methods

- Method is associated with class and not with object, though it can be accessed with object of class as well. e.g. main method of java.
- If static method is accessed with object, compiler does give warning that static methods should be accessed in static way.

## When to Create Static Methods

- When logic is independent of object instance. e.g. computing number of days in a month, to calculate salary. In this case irrespective of which employee the number of days will remain same
- Utility methods e.g. Integer.parseInt()

# Method Overloading

- Allows method with same names but different parameters/different return types with different implementation

# Method Overloading Example

```
public static int ComputeNumberOfDaysInMonth(int monthNumber)
{
    Calendar cal = GregorianCalendar.getInstance();
    cal.set(Calendar.YEAR, 2014);
    cal.set(Calendar.MONTH, monthNumber);

    return cal.getActualMaximum(Calendar.DAY_OF_MONTH);
}

public static int ComputeNumberOfDaysInMonth(String month)
{
    if(month=="January" || month=="March" || month=="May" || month=="July" || month=="Aug" || month=="Oct" || month=="Dec")
        return 31;
    else if(month=="Feb")
        return 28;
    else
        return 30;
}
```

## Method Overloading Output

```
8 //System.out.println(Employee.getYear());
9
10 System.out.println(Employee.ComputeNumberOfDaysInMonth("Feb"));
11 System.out.println(Employee.ComputeNumberOfDaysInMonth(1));
12
```

Console X

```
<terminated> EmployeeObjects [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (31-Dec-2013)
28
31
```

# Constructors

- Methods to instantiate objects
- Properties of constructor
  - Have same name as class
  - Don't have any return value
  - Default constructor is automatically called when an object of class is instantiated(without any parameters)
  - Constructors can be overloaded

# Overloaded Constructors Example

```
 7  private int id;
 8  private String name;
 9  private double salary;
10 private static int YEAR = 2014;
11 public Employee()
12 {
13 }
14
15 public Employee(int id, String name, double salary)
16 {
17     this.id = id;
18     this.name = name;
19     this.salary = salary;
20 }
21
22
23 public void setId(int id)
24 {
25     this.id = id;
26 }
27
```

## This Reference

- Refers to current object instance itself
- Can be used to access instance variables

# Access Modifiers

- **public** access – specifies variables or methods are accessible to anyone, both inside and outside the class and outside the package of class
- Keyword: public

## Public Example

```
public static int YEAR = 2014;

public Employee()
{
}

public Employee(int id, String name, double salary)
{
    this.id = id;
    this.name = name;
    this.salary = salary;
}

public void setId(int id)
{
    this.id = id;
}
```

## Protected Accessibility

- Specifies class members are accessible only to methods of the class and subclass of the class
- Subclass can be in different package
- Keyword: protected

## Protected Example

```
8     protected String name;
9     private double salary;
10    public static int YEAR = 2014;
11
12    protected void setName(String name)
13    {
14        this.name = name;
15    }
16
17    protected void setSalary(double salary)
18    {
19        this.salary = salary;
20    }
21
22
```

## Default Accessibility

- Specifies only classes in the same package will have access to class' variables and methods
- No keyword is specified

## Default Example

```
int id;  
  
void setId(int id)  
{  
    this.id = id;  
}  
  
int getId()  
{  
    return id;  
}  
//uncomment void setName (String name)
```

## Private Access Modifier

- Specifies class members are accessible only within the class
- Keyword: private

## Private Example

```
J Employee.java X
1
2 public class Employee {
3
4     private int id;
5     private String name;|
6     private double salary;
7
8     public Employee()
9     {
10    }
11
12     public int getId()
13     {
14         return id;
15     }
16
17     public String getName()
18     {
19         return name;
20     }
21
22     public double getSalary()
23     {
24         return salary;
25     }
26
27
28
29 }
```

## Coding guidelines

- Instance variables should be declared private and accessor and mutator methods should be provided for these variables

## Exercise

- Define a class Bank, it should have attributes number of saving accounts, address, and number of employees. These fields should be private. Implement the getter and setter methods of these attributes. Implement 2 constructors. One default constructor and the other that takes all the parameters and initializes the above attributes.
- Implement another class that has a main method and which instantiates two objects pof Bank class
  - First object(name the object USBankBranch), and has the attributes saving Number Of SavingAccounts =2000, address="Ma, Boston", numberOfEmployees=4000
  - Second Object(name the object FrBankBranch) and has the attributes Number of saving accounts=5, address="Paris, Fr", number of employees=5000
- Implement a method in the above class that calculates popularity of the bank with the formula  $\text{popularity} = 2 * \text{numberofSavingAccounts} + 3 * \text{numberOfEmployees}$
- Calculate and display the popularity of both the bank objects
- Compare the popularity and display the bank with higher popularity

# Solution

```
public class bank {  
    private int total_accounts;  
    private String address;  
    private int employees;  
  
    public bank(){  
    }  
    public bank(int acc, String add, int emp){  
        this.total_accounts = acc;  
        this.address = add;  
        this.employees = emp;  
    }  
    public void setAccounts(int n){  
        this.total_accounts = n;  
    }  
    public void setAddress(String n){  
        this.address = n;  
    }  
    public void setEmployees(int n){  
        this.employees = n;  
    }  
  
    public int getAccounts(){  
        return this.total_accounts;  
    }  
    public String getAddress(){  
        return this.address;  
    }  
    public int getEmployees(){  
        return this.employees;  
    }  
}
```

## Solution Contd.

```
public class run {
    public static int popularity(bank b){
        int pop;
        pop = 2*b.getAccounts() + 3*b.getEmployees();
        return pop;
    }
    public static void main(String args[]){
        bank USBankBranch = new bank(2000, "MA,Boston", 4000);
        bank FrBankBranch = new bank(5, "Paris,Fr", 5000);

        System.out.format("USBankBranch popularity : %d\n",popularity(USBankBranch));
        System.out.format("FrBankBranch popularity : %d\n",popularity(FrBankBranch));

        if(popularity(USBankBranch) > popularity(FrBankBranch))
            System.out.println("Most popular bank is : USBankBranch");
        else
            System.out.println("Most popular bank is : FrBankBranch");
    }
}
```

# Packages and ClassPath

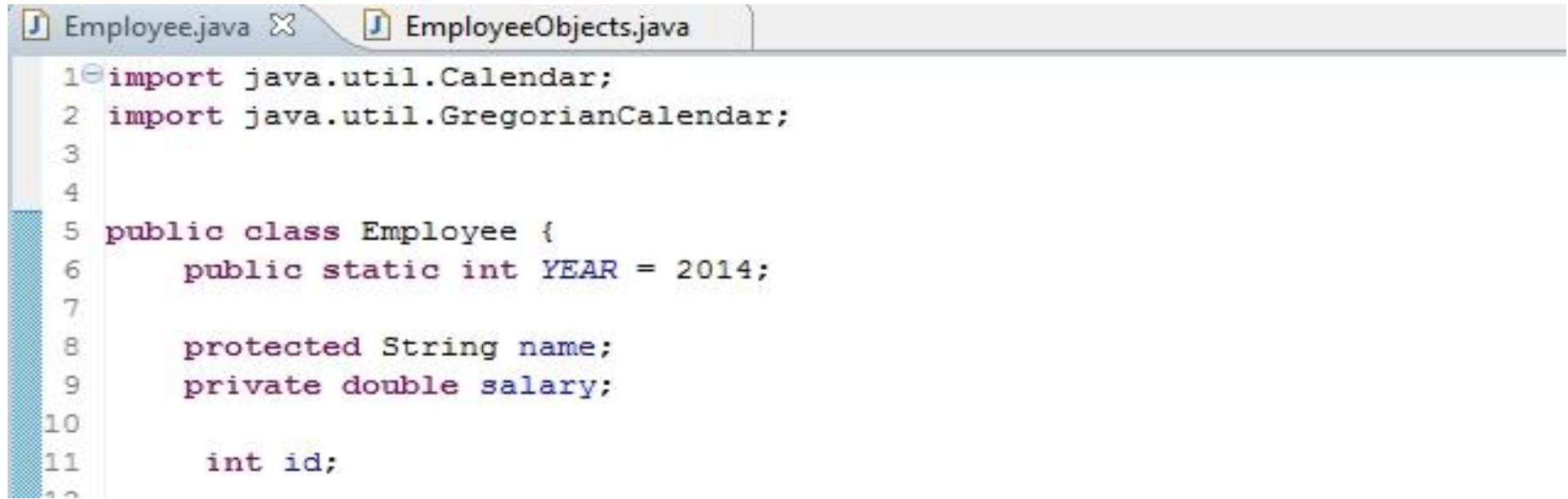
# What are packages?

- Grouping of related classes in a single namespace
- Benefits
  - Programmers can easily identify the related classes and interfaces
  - Classes with the same name can be placed in different packages
  - Access can be restricted by placing classes in appropriate packages

# Importing Classes

- To use classes of other packages import statements can be used
- By default, all java programs import `java.lang*` classes(e.g `String, Integer`)

# Using Classes of Other Packages



```
Employee.java EmployeeObjects.java

1 import java.util.Calendar;
2 import java.util.GregorianCalendar;
3
4
5 public class Employee {
6     public static int YEAR = 2014;
7
8     protected String name;
9     private double salary;
10
11     int id;
```

# Inheritance

# What is Inheritance

- Child class inheriting attributes and methods defined in parent class
- Parent class is called super class
- Child class is called sub class

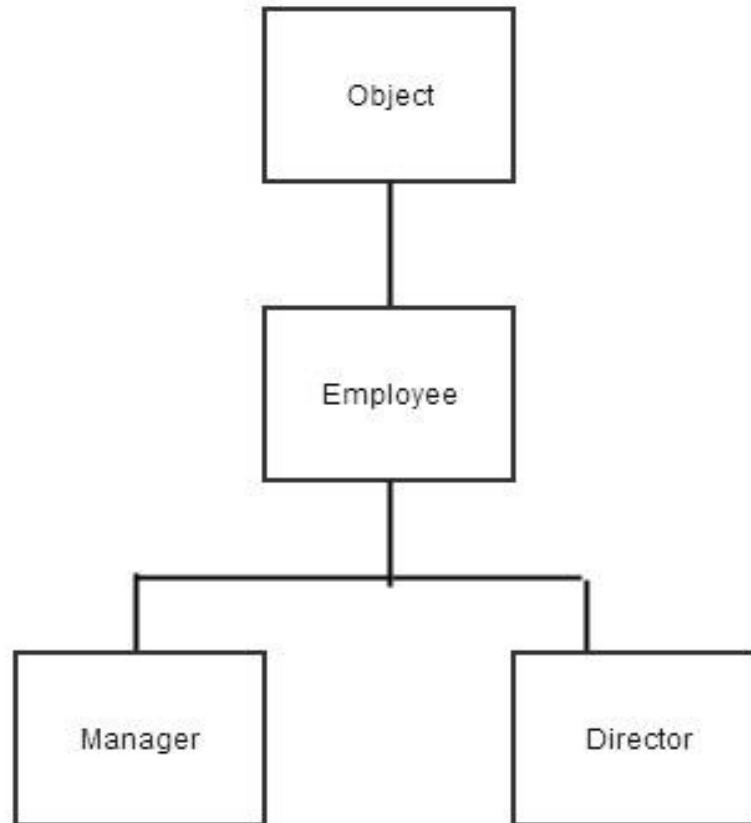
## Why Inheritance

- Reusability- A method (behavior) can be defined only once in parent class, the child class need not implement the same behavior it can just inherit it from parent class
- A subclass only needs to implement differences between itself and parent

# Object Class

- By default all classes have super class as Object class
- Object class doesn't have any parent class
- Defines and implements behavior common to all
  - `getClass()`
  - `equals()`
  - `toString()`
  - ..

# Class Hierarchy



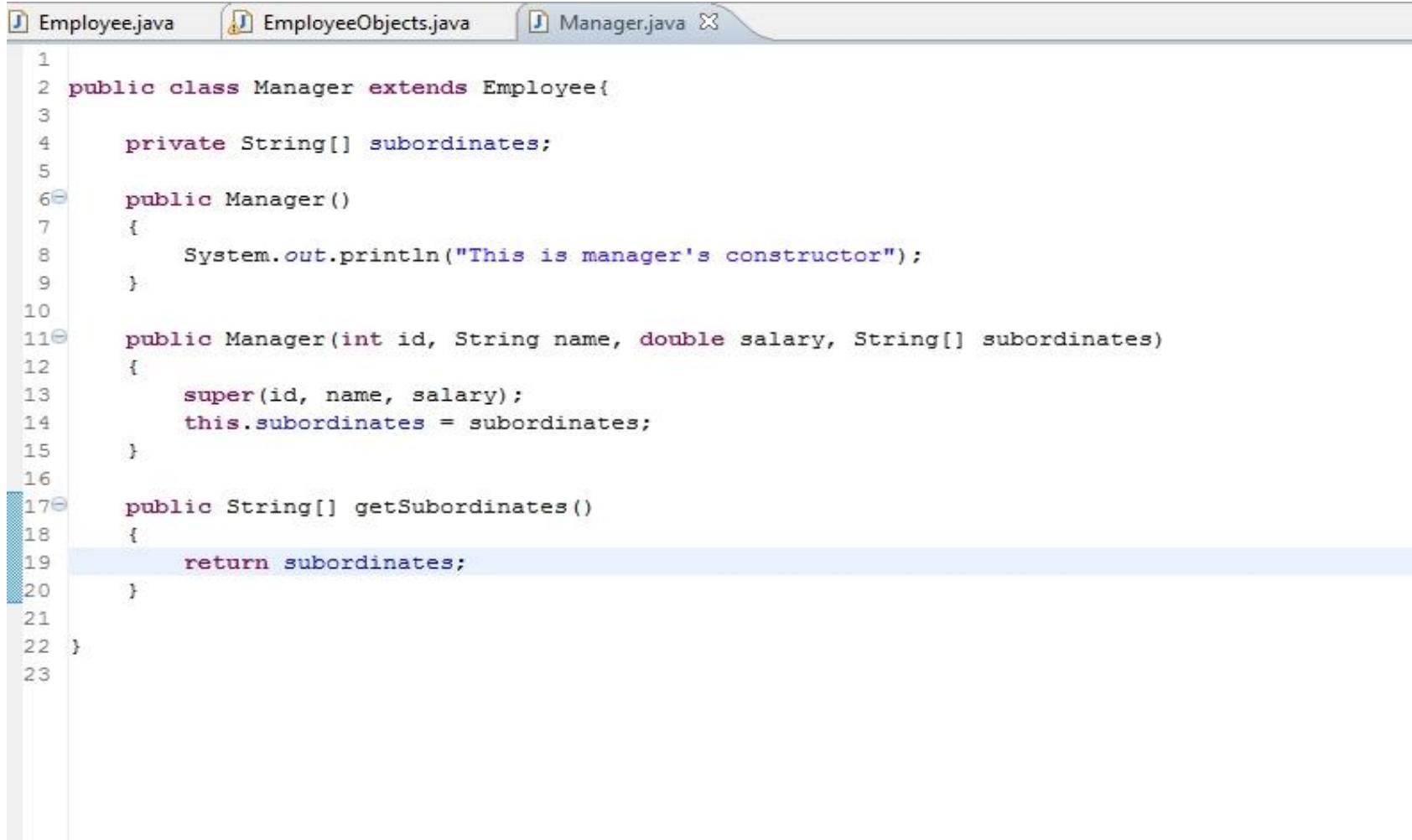
## How to Derive a Subclass

- To derive use extends keyword
- Parent class in Employee, subclass is Manager
- A subclass inherits all the public and protected members
- New Fields can be defined in the subclass that are not in super class
- Methods can also be overridden in subclass

## How Constructor is Called

- A subclass constructor invokes parent's class constructor implicitly
- When Manager class is instantiated, constructor of Employee is invoked implicitly before itself
- A subclass can invoke the constructor of super class explicitly by using super keyword
  - Used for passing parameters

# Manager Subclass



```
Employee.java EmployeeObjects.java Manager.java X
1
2 public class Manager extends Employee{
3
4     private String[] subordinates;
5
6     public Manager()
7     {
8         System.out.println("This is manager's constructor");
9     }
10
11     public Manager(int id, String name, double salary, String[] subordinates)
12     {
13         super(id, name, salary);
14         this.subordinates = subordinates;
15     }
16
17     public String[] getSubordinates()
18     {
19         return subordinates;
20     }
21
22 }
23
```

# Manager's Objects

```
Employee.java EmployeeObjects.java Manager.java
```

```
1 public class EmployeeObjects {
2
3     public static void main(String[] args) {
4
5         String[] subEm1 = new String[]{"Roohi", "Jasleen", "PD"};
6         String[] subEm2 = new String[]{"Einstein", "Newton"};
7
8
9         Manager m1 = new Manager(1, "Alan", 45000.0, subEm1);
10        Manager m2 = new Manager(2, "Bob", 55000.0, subEm2);
11
12
13        System.out.println("Manager's name");
14        System.out.println(m1.getName());
15        System.out.println("Subordinates");
16        for(String sub : m1.getSubordinates())
17            System.out.println(sub);
```

Console X

```
<terminated> EmployeeObjects [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (31-Dec-2013 11:04:10 PM)
Manager's name
Alan
Subordinates
Roohi
Jasleen
PD
```

## Overriding Methods

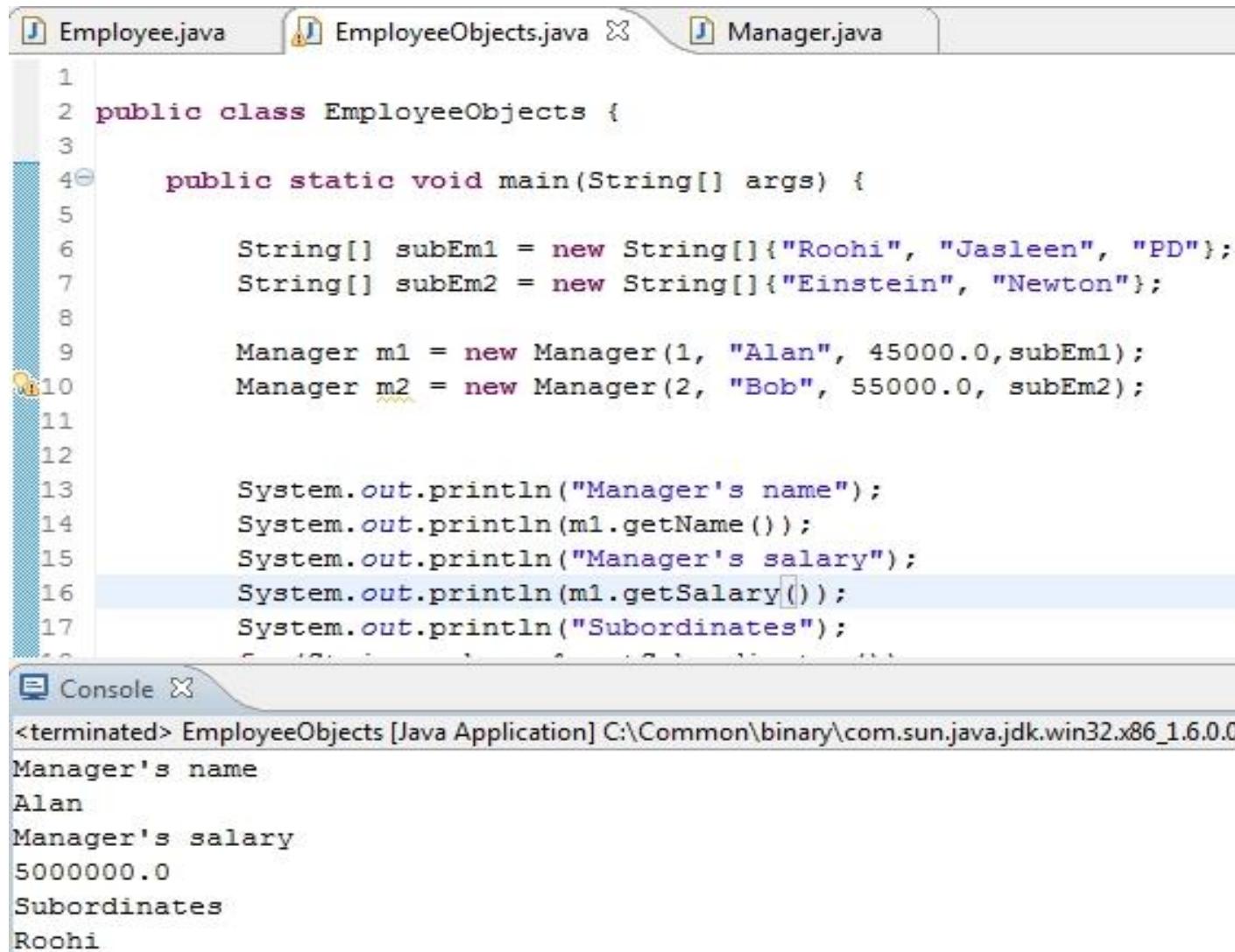
- If a derived class needs to have a different implementation of a certain instance method from that of super class, override that method in subclass
- The overriding method will have same name as the method it overrides
- The overriding method can also return subtype of the type returned by overridden method

## Example

- Let's say manager's salary is fixed 500000\$. The getSalary() method can be overridden in Manager's class to always return this value

```
public class Manager extends Employee{  
  
    private String[] subordinates;  
  
    public Manager()  
    {  
        System.out.println("This is manager's constructor");  
    }  
  
    public Manager(int id, String name, double salary, String[] subordinates)  
    {  
        super(id, name, salary);  
        this.subordinates = subordinates;  
    }  
  
    public String[] getSubordinates()  
    {  
        return subordinates;  
    }  
  
    /**  
     * Overridden method  
     */  
    public double getSalary()  
    {  
        return 5000000.0;  
    }  
}
```

# Overridden Method Invocation



The screenshot shows an IDE interface with three tabs at the top: Employee.java, EmployeeObjects.java, and Manager.java. The EmployeeObjects.java tab is active, displaying the following code:

```
1 public class EmployeeObjects {
2
3     public static void main(String[] args) {
4         String[] subEm1 = new String[]{"Roohi", "Jasleen", "PD"};
5         String[] subEm2 = new String[]{"Einstein", "Newton"};
6
7         Manager m1 = new Manager(1, "Alan", 45000.0, subEm1);
8         Manager m2 = new Manager(2, "Bob", 55000.0, subEm2);
9
10
11
12         System.out.println("Manager's name");
13         System.out.println(m1.getName());
14         System.out.println("Manager's salary");
15         System.out.println(m1.getSalary());
16         System.out.println("Subordinates");
17 }
```

The line `System.out.println(m1.getSalary());` is highlighted with a light blue background. Below the code editor is a console window titled "Console". The output from the console is:

```
<terminated> EmployeeObjects [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.0
Manager's name
Alan
Manager's salary
5000000.0
Subordinates
Roohi
```

## Access Modifier of Overridden Methods

- The access specifier for an overriding method can allow more, but not less, access than the overridden method

# Final Class

- Final classes
  - Classes that can't be extended are declared final
  - Wrapper classes
  - String class

# Final Methods

- Methods that cannot be overridden
- To declare final methods
  - `public final [return type][method name]([parameters]) {  
}`

## Exercises

- Using the fundamentals described in the session design framework for calculating the salary and displaying name, id and address for each employee, manager and director for company Moogle Inc. The monthly salary of director is  $60,000 * \text{number of days in month}$ , salary of manager is  $50,000 * \text{number of days in a month}$  and for employee it's  $20,000 * \text{number of days in a month}$ (hint use employee base class and Director and Manager extend Employee, define the attributes using principles of encapsulation)
- Additionally Director has a list of departments and Manager has a list of subordinates, modify the above class accordingly.
- Define another class that instantiates 2 directors(assume address of each employee is IND)
  - Steven (department = "IT", "Admin") id = 1
  - Dale(department = "finance", "Engg") id =2

Instantiate 4 managers

- Jao (id =3, subordinates = "John, id =7", "Susan, id=8")
  - Ross(id=4, subordinates="sam id =9","Chris, id =10")
  - Joe (id=5, subordinates="Monica, id=11")
  - Tracy(id=6, subordinates None)
- Display the list of all employees their subordinates, salary, id, department and address). Where attribute is not applicable display -

# Solution

```

public class employee {
    private static int days_month = 30;
    protected String name;
    protected int id;
    protected String address;
    protected float salary;

    public employee(String name, int id) {
        this.name = name;
        this.id = id;
        this.address = "IND";
        this.salary = 20000*days_month;
    }
    public void getDetails(){
        System.out.println('\n');
        System.out.println("Name : "+name);
        System.out.println("ID : "+id);
        System.out.println("Address : "+address);
        System.out.println("Salary : "+salary);
    }
}

```

```

public class manager extends employee {
    String[] sub_name;
    int[] sub_id;

    public manager(String name,int id,String[] sub_name,int[] sub_id){
        super(name,id);
        this.sub_name = sub_name;
        this.salary = 5*this.salary/2;
        this.sub_id = sub_id;
    }
    public void getDetails(){
        System.out.println('\n');
        System.out.println("Manager Name : "+name);
        System.out.println("ID : "+id);
        System.out.println("Address : "+address);
        System.out.println("Salary : "+salary);
        if(sub_name!=null){
            System.out.println('\n'+ "---Details of subordinates---");
            int l = sub_name.length;
            int i=0;
            while ( i < l){
                employee e = new employee(sub_name[i],sub_id[i]);
                e.getDetails();
                i++;
            }
        }
    }
}

```

## Solution Cont.

```

public class director extends employee {
    String[] dept;

    public director(String name,int id,String[] dept){
        super(name,id);
        this.dept = dept;
        this.salary = 6*this.salary/2;
    }
    public void getDetails(){
        System.out.println('\n');
        System.out.println("Director Name : "+name);
        System.out.println("ID : "+id);
        System.out.println("Address : "+address);
        System.out.println("Salary : "+salary);
        int i=1;
        int l = dept.length;
        String depts = dept[0] ;
        while( i<l ){
            depts= depts+","+dept[i];
            i++;
        }
        System.out.println("Departments : "+depts);
    }
}

public class run {

    public static void main(String[] args) {
        String[] sub1={"John","Susan"};
        int[] id1={7,8};
        String[] sub2={"Sam","Chris"};
        int[] id2={9,10};
        String[] sub3={"Monica"};
        int[] id3={11};
        String[] sub4=null;
        int[] id4=null;

        String[] dept1={"IT","Admin"};
        String[] dept2={"Finance","Engg"};

        director dir1 = new director("Steven",1,dept1);
        director dir2 = new director("Dale",2,dept2);

        manager man1 = new manager("Jao",3,sub1,id1);
        manager man2 = new manager("Ross",4,sub2,id2);
        manager man3 = new manager("Joey",5,sub3,id3);
        manager man4 = new manager("Tracy",6,sub4,id4);

        dir1.getDetails();
        dir2.getDetails();
        man1.getDetails();
        man2.getDetails();
        man3.getDetails();
        man4.getDetails();
    }
}

```

# Abstract Class and Interface

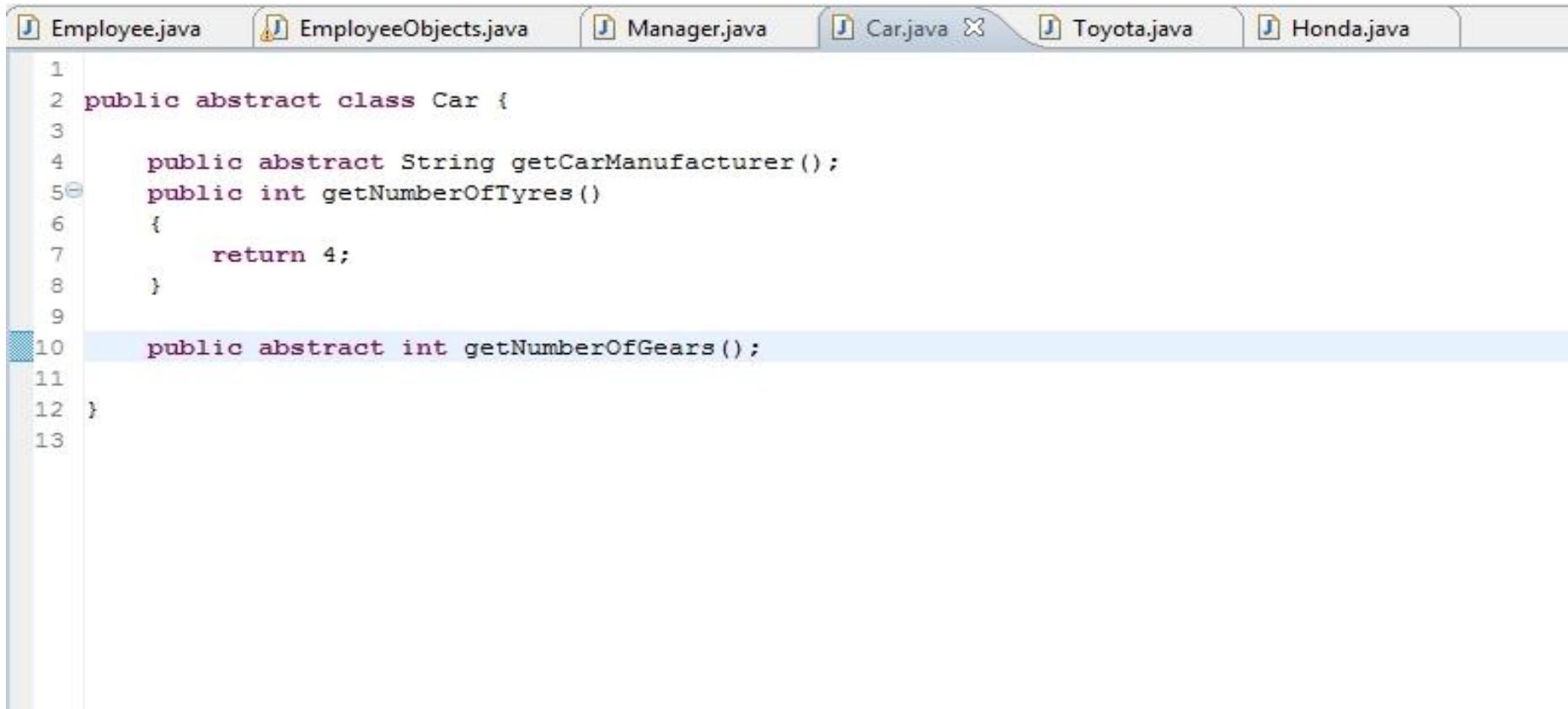
## Abstract Method

- Method doesn't have implementation
- Abstract methods don't have body
- `public abstract void someAbstractMethod();`

## Abstract Class

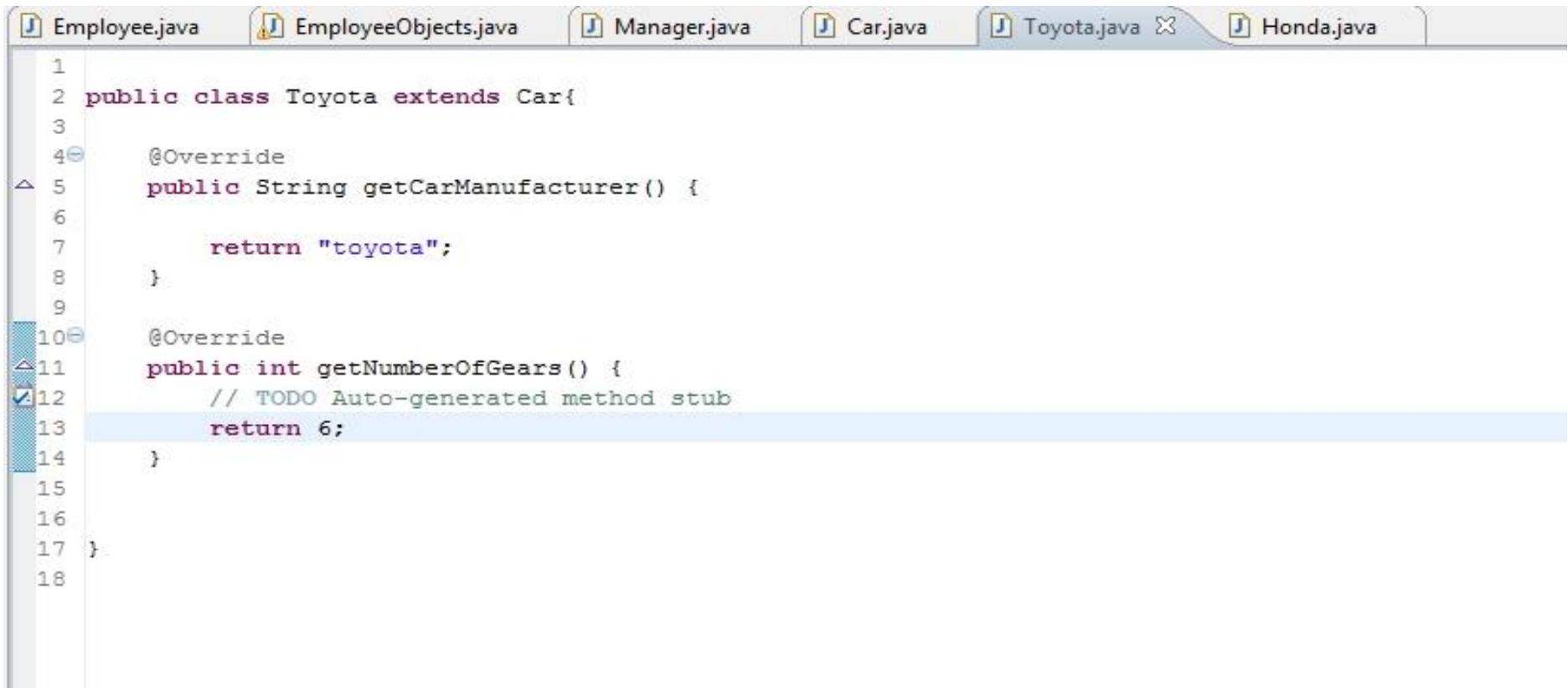
- It contains at least one or more abstract methods
- Abstract class can't be instantitated
- Another concrete class has to be created to provide implementation of all abstract methods
- The concrete class uses extends keyword

# Abstract Class



```
Employee.java EmployeeObjects.java Manager.java Car.java ✘ Toyota.java Honda.java
1
2 public abstract class Car {
3
4     public abstract String getCarManufacturer();
5     public int getNumberOfTyres()
6     {
7         return 4;
8     }
9
10    public abstract int getNumberOfGears();
11
12 }
```

# Subclass



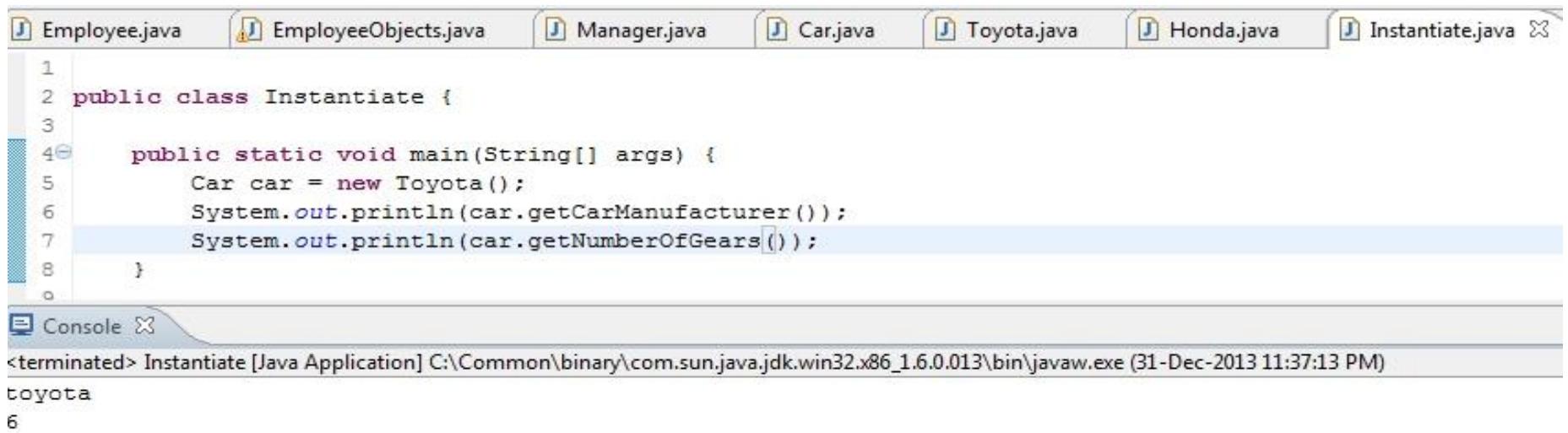
The screenshot shows a Java code editor with several tabs at the top: Employee.java, EmployeeObjects.java, Manager.java, Car.java, Toyota.java (selected), and Honda.java. The code in the Toyota.java tab is as follows:

```
1
2 public class Toyota extends Car{
3
4     @Override
5     public String getCarManufacturer() {
6
7         return "toyota";
8     }
9
10    @Override
11    public int getNumberOfGears() {
12        // TODO Auto-generated method stub
13        return 6;
14    }
15
16
17 }
18
```

## Contd...

```
Employee.java EmployeeObjects.java Manager.java Car.java Toyota.java Honda.java X
1
2 public class Honda extends Car {
3
4     @Override
5     public String getCarManufacturer() {
6         // TODO Auto-generated method stub
7         return "Honda";
8     }
9
10    @Override
11    public int getNumberOfGears() {
12        // TODO Auto-generated method stub
13        return 5;
14    }
15
16 }
17
```

# Instantiation



The screenshot shows a Java development environment with several tabs at the top: Employee.java, EmployeeObjects.java, Manager.java, Car.java, Toyota.java, Honda.java, and Instantiate.java. The Instantiate.java tab is active, displaying the following code:

```
1 public class Instantiate {
2
3     public static void main(String[] args) {
4         Car car = new Toyota();
5         System.out.println(car.getCarManufacturer());
6         System.out.println(car.getNumberOfGears());
7     }
8 }
```

Below the code editor is a 'Console' tab showing the application's output:

```
<terminated> Instantiate [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (31-Dec-2013 11:37:13 PM)
toyota
6
```

## When to Use Abstract Class

- Abstract methods are used when the subclasses have same behavior but different implementation. (This is polymorphism)
- Use abstract classes to define broad types of behavior on top of object-oriented programming class hierarchy, and use its subclasses to provide implementation details of abstract class

# Interface

- All methods in the interface are abstract implicitly
- Defines signature of a set of methods, without the body
- A class implementing interface must implement all the methods defined in interface
- Use interface modifier to the class declaration

# Use of interface

- It defines way of specifying behavior of classes

```

Employee.java EmployeeObjects.java Manager.java Car.java Toyota.java Honda.java Instantiate.java Polygon.java Triangle.java

1 public interface Polygon {
2
3     public int getNumberOfSides();
4     public String getName();
5     public float AngleInDegrees();
6
7 }
8
9

```

```

Employee.java EmployeeObjects.java Manager.java Car.java Toyota.java Honda.java Instantiate.java Polygon.java Triangle.java

1 public class Triangle implements Polygon {
2
3     @Override
4     public float AngleInDegrees() {
5
6         return 60;
7     }
8
9
10    @Override
11    public String getName() {
12        // TODO Auto-generated method stub
13        return "Triangle";
14    }
15
16    @Override
17    public int getNumberOfSides() {
18        // TODO Auto-generated method stub
19        return 3;
20    }
21
22 }

```

## Contd...

```
Employee.java EmployeeObjects.java Manager.java Car.java Toyota.java Honda.java Instantiate.java Polygon.java Triangle.java Square.java
1 public class Square implements Polygon{
2
3     @Override
4     public float AngleInDegrees() {
5         // TODO Auto-generated method stub
6         return 90;
7     }
8
9
10    @Override
11    public String getName() {
12        // TODO Auto-generated method stub
13        return "Square";
14    }
15
16    @Override
17    public int getNumberOfSides() {
18        // TODO Auto-generated method stub
19        return 4;
20    }
21
22 }
```

## Why Interface?

- To give API without giving implementation to the client
- This is the concept of encapsulation
- The implementation can change without affecting the user of the interface
- The caller doesn't need implementation only interface is needed
- A class can implement multiple interfaces, and hence multiple behaviors can be defined for it

```
Employee.java EmployeeObjects.java Car.java Toyota.java Honda.java Instantiate.java Polygon.java Triangle.java Square.java InterfaceUse.java X
1
2 public class InterfaceUse {
3
4     public static void main(String[] args) {
5         Polygon newPolygon = new Triangle();
6         System.out.println(newPolygon.getName());
7         System.out.println(newPolygon.getNumberOfSides());
8     }
}
Console X
<terminated> InterfaceUse [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (01-Jan-2014 12:01:08 AM)
Triangle
3
```

## Interface vs Abstract Class

- All methods of interface are abstract, however abstract class can have methods which are implemented
- Default implementation can be defined in abstract class
- Interfaces can only define constants whereas abstract classes can have fields

# Java Exception Handling

# What is An Exception

- Represents an exception event – typically an error
- Exception examples
  - Divide by zero
  - Accessing array elements beyond it's range
  - Invalid input
  - Out of memory

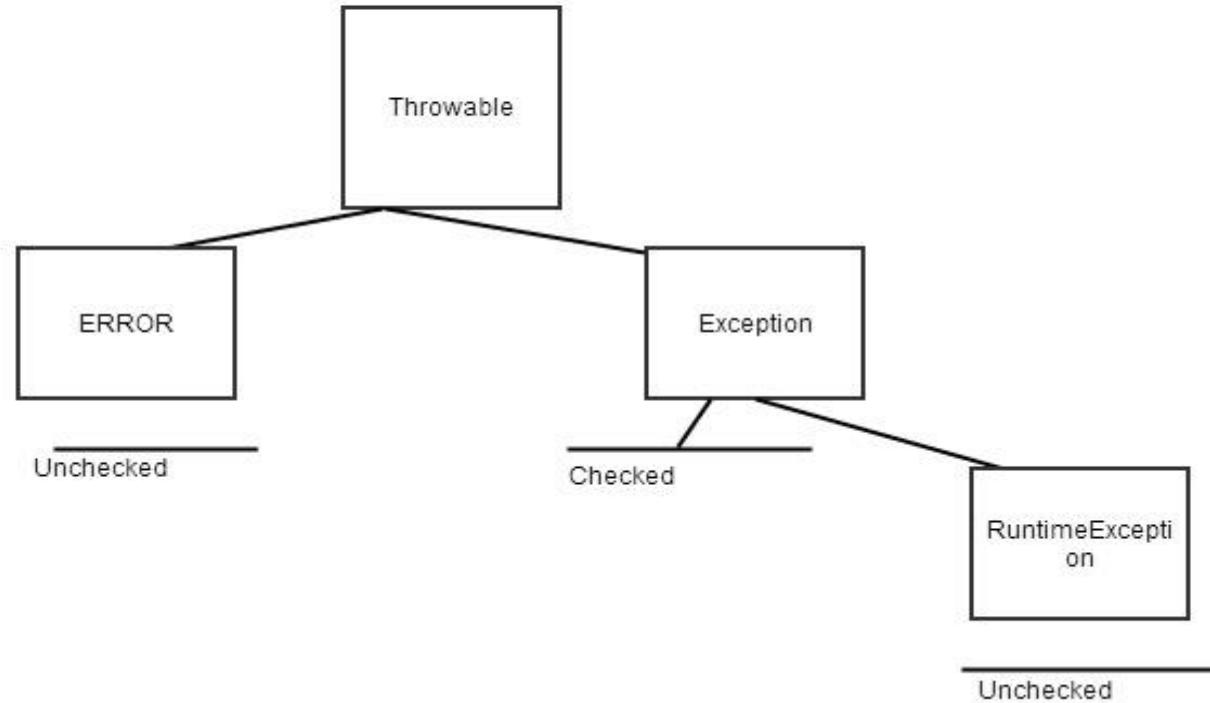
# Checked Exception

- Exceptions checked at compile time.

```
■ class Main {  
■     public static void main(String[] args) throws IOException {  
■         FileReader file = new FileReader("C:\\test\\pristine.txt");  
■         BufferedReader fileInput = new BufferedReader(file);  
  
■         // Print first 10 lines of file "C:\\test\\pristine.txt"  
■         for (int counter = 0; counter < 10; counter++)  
■             System.out.println(fileInput.readLine());  
  
■         fileInput.close();  
■     }  
■ }
```

## Unchecked Exception

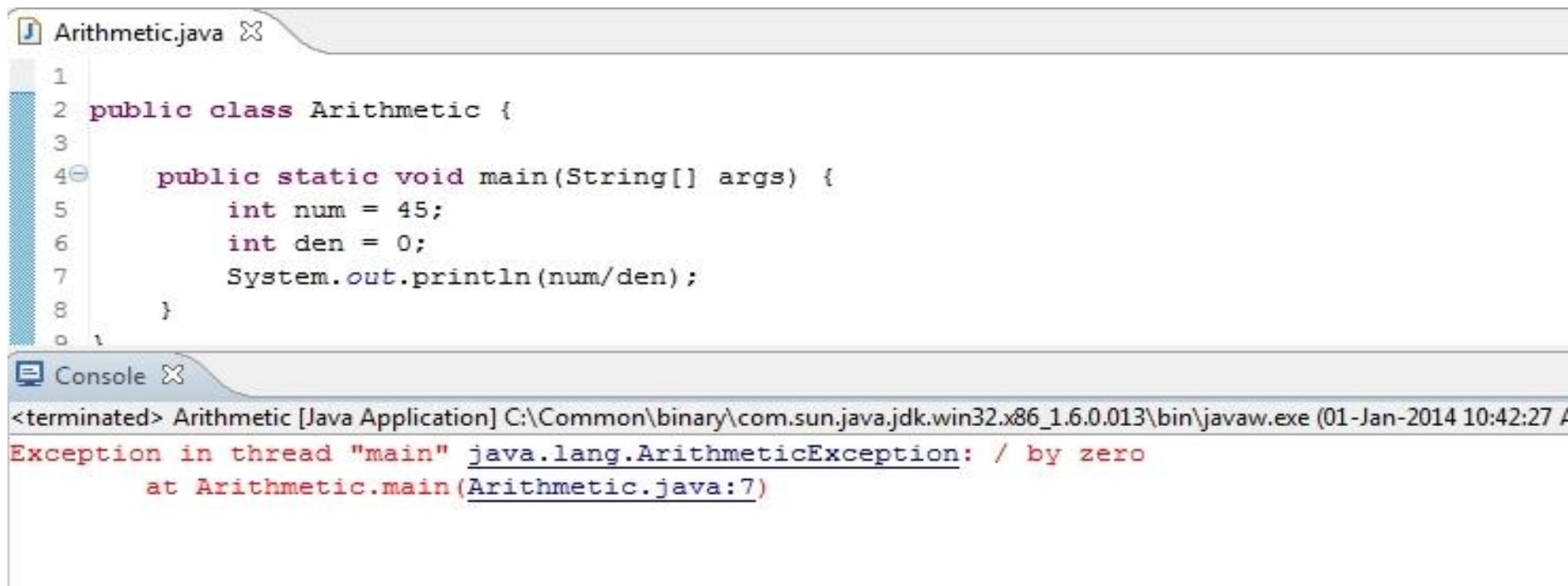
- Exceptions not checked at compile time, but occur at runtime



## Rules of Exception

- A method is required to either catch or list all exceptions it might throw except error or RuntimeException or their subclasses

# RunTimeException



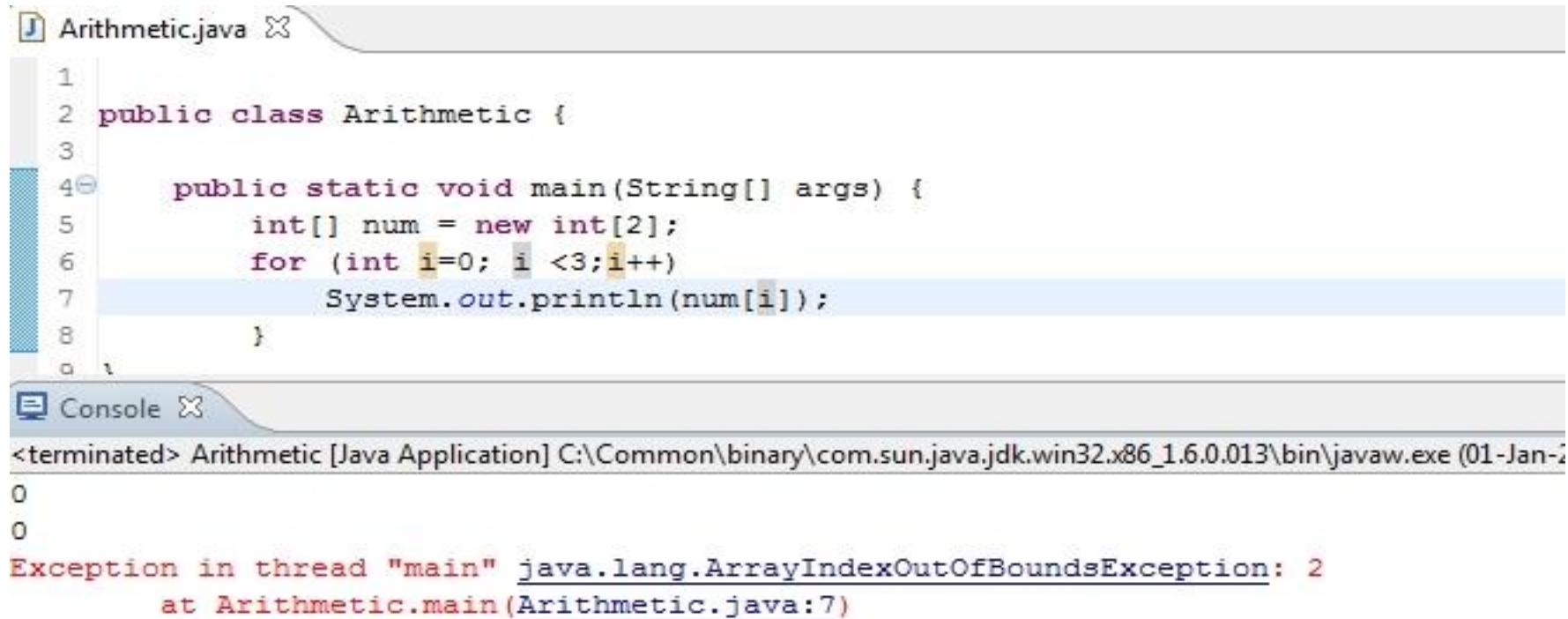
The screenshot shows a Java development environment with two tabs: 'Arithmetic.java' and 'Console'. The code in 'Arithmetic.java' is as follows:

```
1
2 public class Arithmetic {
3
4     public static void main(String[] args) {
5         int num = 45;
6         int den = 0;
7         System.out.println(num/den);
8     }
9 }
```

The 'Console' tab displays the output of running the application:

```
<terminated> Arithmetic [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (01-Jan-2014 10:42:27 A
Exception in thread "main" java.lang.ArithmetricException: / by zero
at Arithmetic.main(Arithmetic.java:7)
```

# RunTimeException



The screenshot shows a Java development environment with two tabs: "Arithmetic.java" and "Console".

**Arithmetic.java:**

```
1
2 public class Arithmetic {
3
4     public static void main(String[] args) {
5         int[] num = new int[2];
6         for (int i=0; i <3;i++)
7             System.out.println(num[i]);
8     }
9 }
```

**Console:**

```
<terminated> Arithmetic [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (01-Jan-2012)
0
0
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
at Arithmetic.main(Arithmetic.java:7)
```

# Catching Exceptions

- Syntax

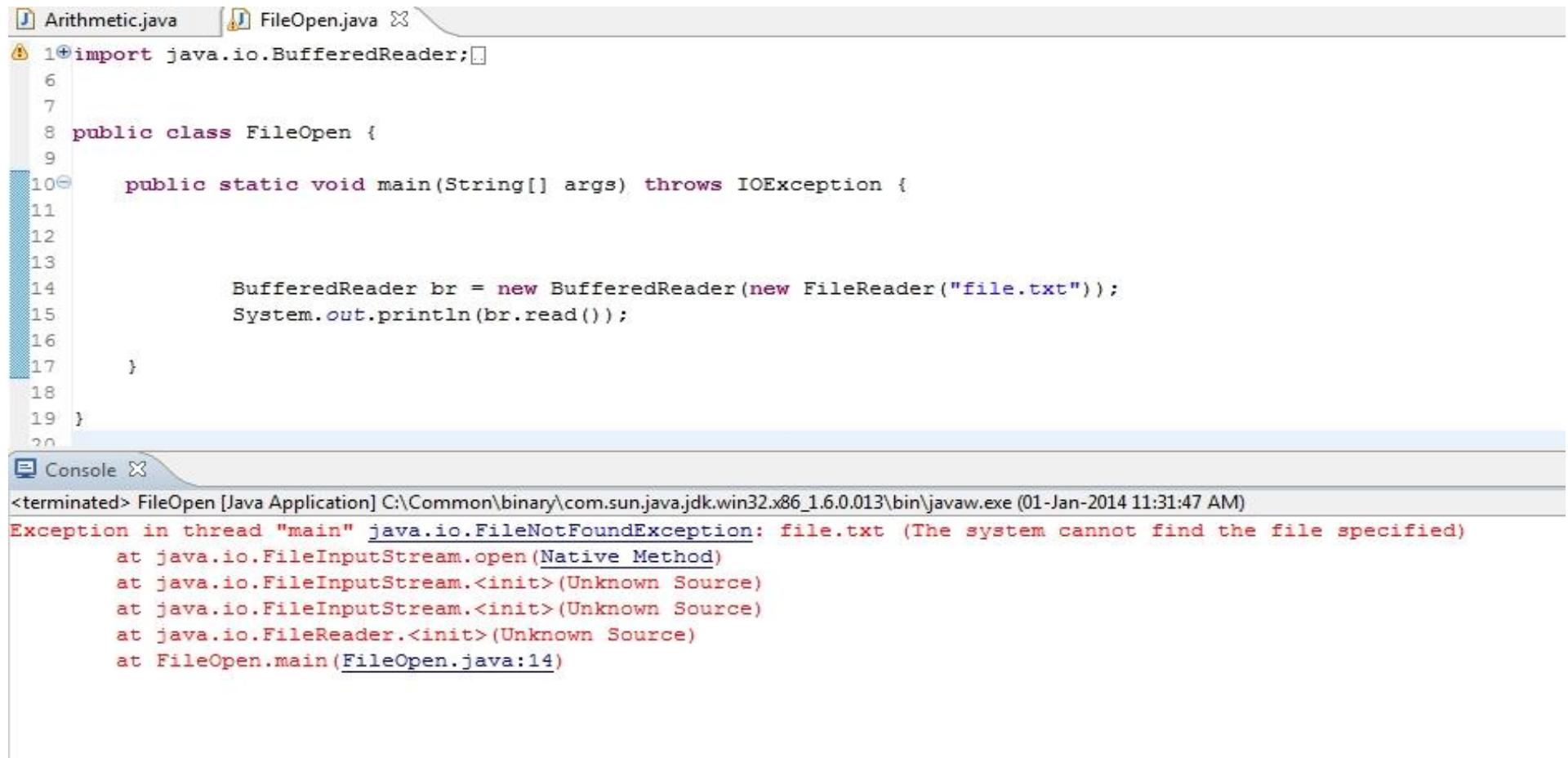
```
try{  
    <code to be monitored for exceptions>  
}catch(<ExceptionType1> <objName>){  
    <handler if Exception1 occurs>  
}  
  
...  
  
}catch(<Exception Typen><objName>){  
    <handler if Exception N occurs>  
}
```

```
Arithmetic.java ✘ FileOpen.java ✘
1+import java.io.BufferedReader;□
6
7
8 public class FileOpen {
9
10    public static void main(String[] args) {
11
12
13        try {
14            BufferedReader br = new BufferedReader(new FileReader("file.txt"));
15            System.out.println(br.read());
16        } catch (FileNotFoundException e) {
17            e.printStackTrace();
18        } catch (IOException e)
19        {
20            e.printStackTrace();
21        }
22    }
}
```

```
Console ✘
<terminated> FileOpen [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (01-Jan-2014 11:45:00)
java.io.FileNotFoundException: file.txt (The system cannot find the file specified)
    at java.io.FileInputStream.open(Native Method)
    at java.io.FileInputStream.<init>(Unknown Source)
    at java.io.FileInputStream.<init>(Unknown Source)
    at java.io.FileReader.<init>(Unknown Source)
    at FileOpen.main(FileOpen.java:14)
```

## Throws Clause

- If a method may cause an exception to occur but doesn't catch it, then it must say so using throws clause



The screenshot shows an IDE interface with two tabs at the top: "Arithmetic.java" and "FileOpen.java". The "FileOpen.java" tab is active, displaying the following code:

```
1+import java.io.BufferedReader;□
2
3
4 public class FileOpen {
5
6     public static void main(String[] args) throws IOException {
7
8         BufferedReader br = new BufferedReader(new FileReader("file.txt"));
9         System.out.println(br.read());
10    }
11
12
13
14
15
16
17
18
19 }
20
```

Below the code editor is a "Console" tab showing the execution of the program:

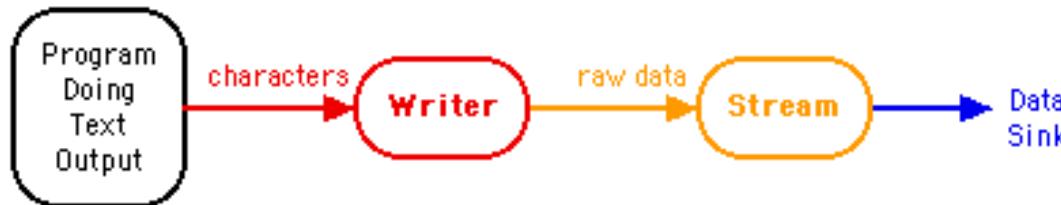
```
<terminated> FileOpen [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013\bin\javaw.exe (01-Jan-2014 11:31:47 AM)
Exception in thread "main" java.io.FileNotFoundException: file.txt (The system cannot find the file specified)
at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.<init>(Unknown Source)
at java.io.FileInputStream.<init>(Unknown Source)
at java.io.FileReader.<init>(Unknown Source)
at FileOpen.main(FileOpen.java:14)
```

# Input / Output options in JAVA

# Data Flow



Input with a Reader



Output with a Writer

# Input from Keyboard ( through console )

Keyboard :

## 1) Through Buffered Reader

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String s = br.readLine();
```

## 2) Through Scanner

```
import java.util.Scanner;

Scanner sc = new Scanner(System.in);
String s = sc.nextLine();
```

# Input From File

```
package file;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class read {
    public static void main(String[] args) throws IOException{
        String path = "C://Users/Pankaj/Desktop/abc.txt";
        File f = new File(path);
        BufferedReader br = new BufferedReader(new FileReader(f));

        String s = br.readLine();
        System.out.println("1st Line");
        System.out.println(s);

        s = br.readLine();
        System.out.println("2nd Line");
        System.out.println(s);

        br.close();
    }
}
```

```
<terminated> read [Java Application] C:\Program Files (x8b)\Java\jre\bin\javaw.exe (Feb 11, 2014, 3:44:02 PM)
1st Line
Contents of this file are to be read through a java program and then shown in console.
2nd Line
This is second line of the file.
```

# Output

## 1) To console

As already seen, output to console can be through many commands such as

```
System.out.println();
```

```
System.out.format();
```

## 2) To File

Just like we used BufferedReader and FileReader to read data from a file, we can use BufferedWriter and FileWriter to write to file.

\*Stream of bytes can also be written using OutputStream

\*Tokenizer can be used to break a string line into tokens with specified delimiters.

# Output to File

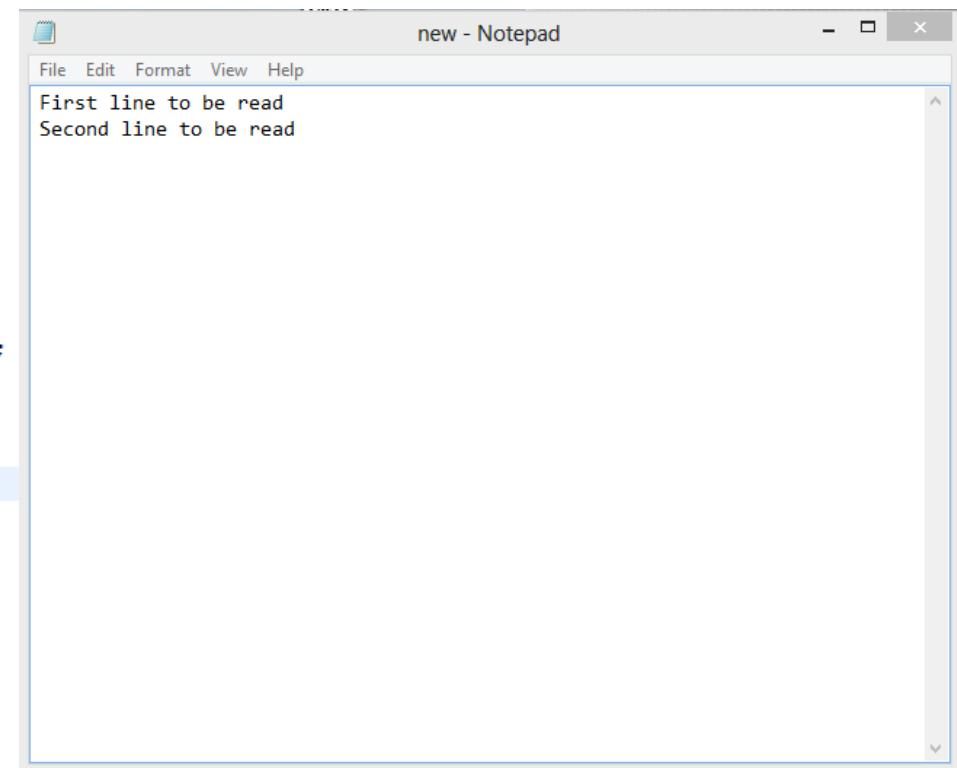
```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class write {
    public static void main(String[] args) throws IOException{
        String path = "C://Users/Pankaj/Desktop/new.txt";
        File f = new File(path);
        BufferedWriter br = new BufferedWriter(new FileWriter(f));

        String s = "First line to be read";
        br.write(s);
        br.newLine();

        s= "Second line to be read";
        br.write(s);

        br.close();
    }
}
```



# Control Structure

# Types

- Repetition
  - Allows to execute specific section number of times
- Decision Control Structure
  - Allows to select specific code to be executed

## Decision Control

- if statement
- if else statement
- if else if statement

# if Statement

- if(boolean expression true)  
execute following statements

```
-----  
if(m1.getName ()=="Alan")  
{  
    for(String sub : ((Manager) m1).getSubordinates())  
        System.out.println(sub);  
}  
  
}
```

## if else

- if (boolean expression true)
  - execute following statements
- else
  - execute these statements

```
if(m1.getName ()=="Alan")
{
    for(String sub : ((Manager) m1).getSubordinates())
        System.out.println(sub);
}

else
{
    System.out.println("You can view only Alan's subordinates");
}
```

## if else else if

- if (boolean statement is true)
  - execute these statements
- else if(this expression is true)
  - execute these statements
- else
  - execute these statements

## if else if else

```
public class EmployeeObjects {  
  
    public static void main(String[] args) {  
  
        String[] subEm1 = new String[]{"Roohi", "Jasleen", "PD"};  
        String[] subEm2 = new String[]{"Einstein", "Newton"};  
  
        Employee m1 = new Manager(1, "Alan", 45000.0, subEm1);  
        Manager m2 = new Manager(2, "Bob", 55000.0, subEm2);  
  
        System.out.println("Manager's name");  
        System.out.println(m1.getName());  
        System.out.println("Manager's salary");  
        System.out.println(m1.getSalary());  
        System.out.println("Subordinates");  
  
        if(m2.getSubordinates().length==3)  
            System.out.println("Manager has 3 subordinates");  
        else if (m2.getSubordinates().length==2)  
            System.out.println("Manager has 2 subordinates");  
        else if (m2.getSubordinates().length==4)  
            System.out.println("Manager has 4 subordinates");  
        else  
            System.out.println("Manager has 1 subordinate");  
  
    }  
}
```

# Switch Statement

- Allows branching on multiple outcomes

```
switch(switch_expression){  
    case case_selector1:  
        statement1;  
        statement2;  
    case case_selector2:  
        statement1;  
        statement2;  
    default:  
        statement1;  
        statement2;  
}
```

# Switch Statement

```
Employee m1 = new Manager(1, "Alan", 45000.0,subEm1);
Manager m2 = new Manager(2, "Bob", 55000.0, subEm2);

System.out.println("Manager's name");
System.out.println(m1.getName());
System.out.println("Manager's salary");
System.out.println(m1.getSalary());
System.out.println("Subordinates");

int numberOfSubordinates = m2.getSubordinates().length;

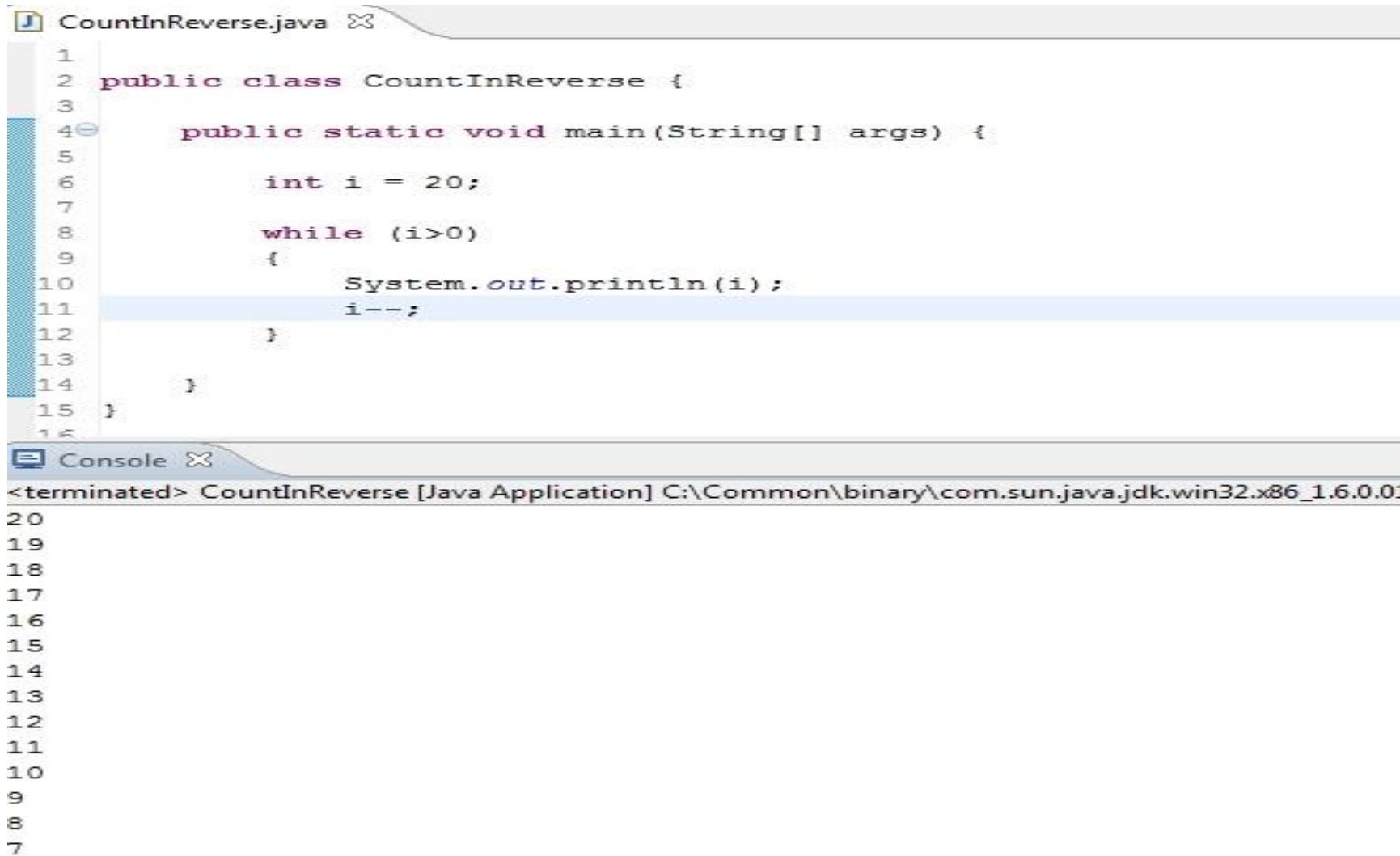
switch(numberOfSubordinates){
case 3:
    System.out.println("Manager has 3 subordinates");
    break;
case 2:
    System.out.println("Manager has 2 subordinates");
    break;
case 4:
    System.out.println("Manager has 4 subordinates");
    break;
default:
    System.out.println("Manager has 1 subordinate");
    break;
}
```

## While Loop(Repetition)

- Is a statement or block of code that is repeated as long as some condition is satisfied
- ```
while(boolean_expression){  
    statement1;  
    statement2;  
}
```

The statements inside while loop get executed as long as boolean\_expression is true

# While Loop



The screenshot shows a Java development environment with two tabs: "CountInReverse.java" and "Console".

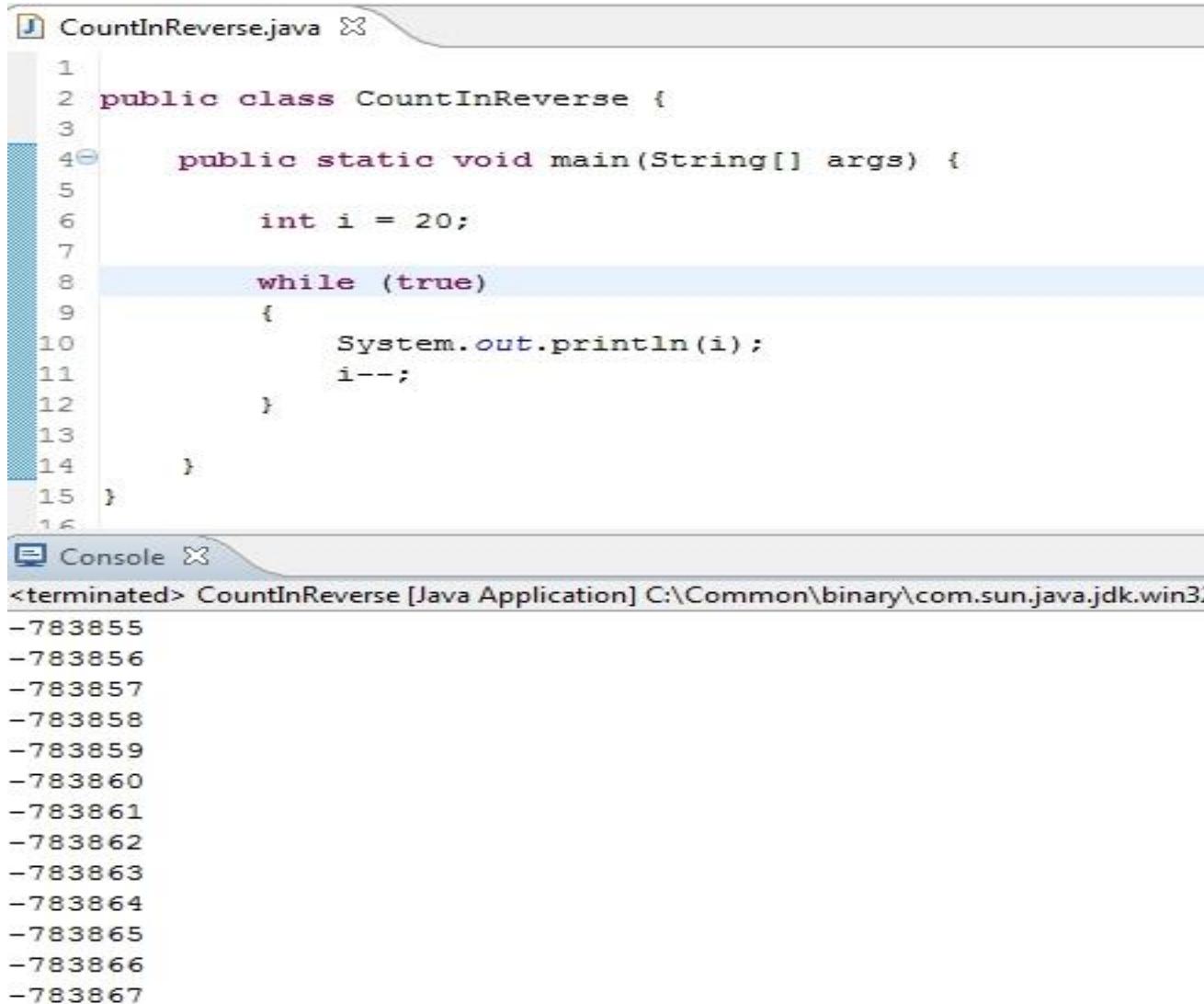
**CountInReverse.java:**

```
1 public class CountInReverse {
2     public static void main(String[] args) {
3         int i = 20;
4         while (i>0)
5         {
6             System.out.println(i);
7             i--;
8         }
9     }
10 }
```

**Console:**

```
<terminated> CountInReverse [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.0:
20
19
18
17
16
15
14
13
12
11
10
9
8
7
```

# Infinite Loop Always True



The screenshot shows a Java IDE interface with two tabs: "CountInReverse.java" and "Console".

The code in "CountInReverse.java" is:

```
1 public class CountInReverse {
2     public static void main(String[] args) {
3         int i = 20;
4         while (true)
5         {
6             System.out.println(i);
7             i--;
8         }
9     }
10 }
```

The "Console" tab shows the output of the program:

```
<terminated> CountInReverse [Java Application] C:\Common\binary\com.sun.java.jdk.win32
-783855
-783856
-783857
-783858
-783859
-783860
-783861
-783862
-783863
-783864
-783865
-783866
-783867
```

## Do While Loop

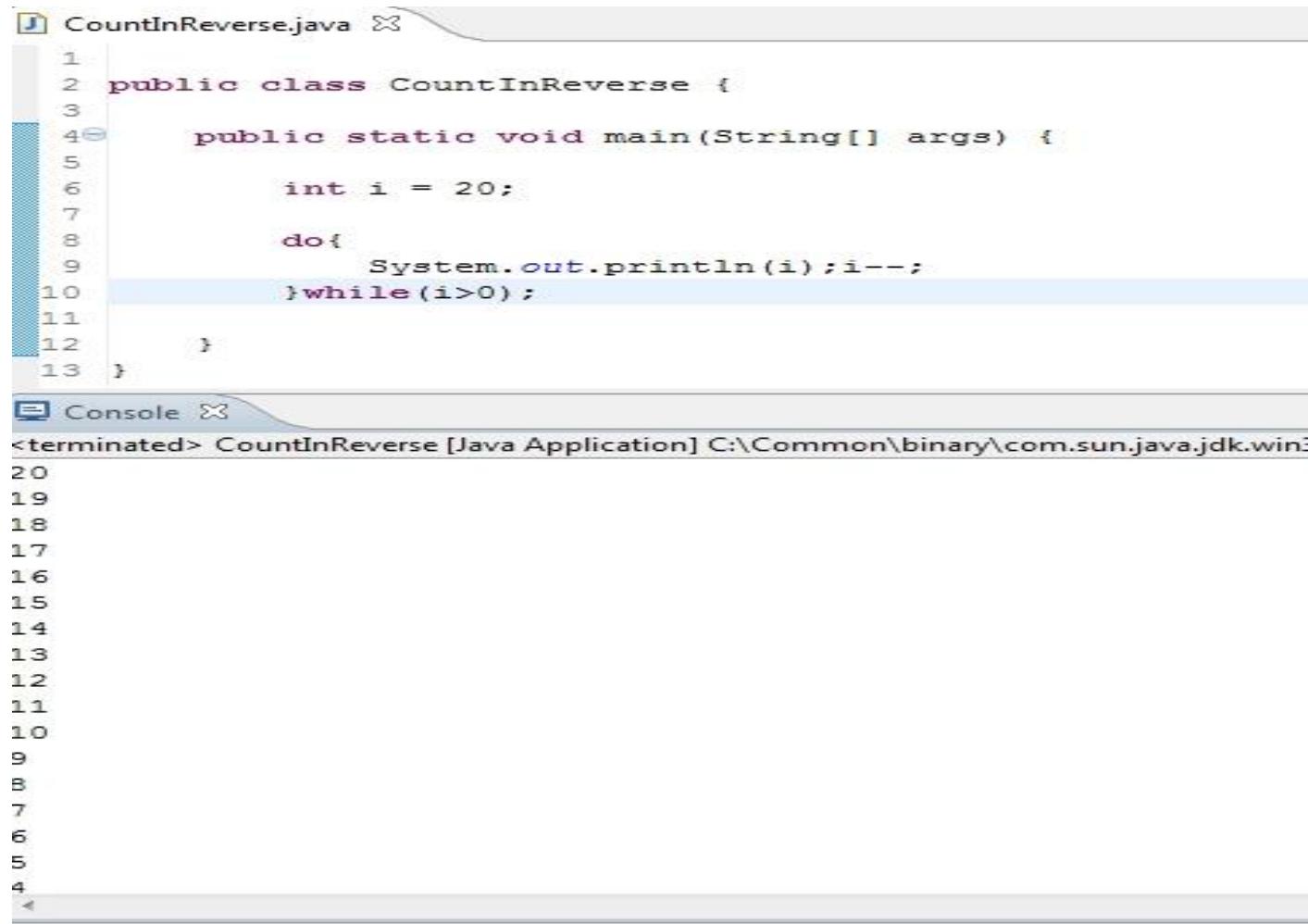
- Similar to while loop, however statements inside do while get executed at least once
- do{

```
statement1;
```

```
statement2;
```

```
}while(boolean_expression);
```

# Do While Loop



The screenshot shows a Java development environment with two windows:

- CountInReverse.java**: This window displays the Java code for a program that counts down from 20 to 1. The code uses a do-while loop. Line 10, which contains the loop condition, is highlighted in light blue.
- Console**: This window shows the output of the program. It prints the numbers 20 through 1, each on a new line, demonstrating the execution of the do-while loop.

```
1 public class CountInReverse {
2     public static void main(String[] args) {
3         int i = 20;
4         do{
5             System.out.println(i);i--;
6         }while(i>0);
7     }
8 }
```

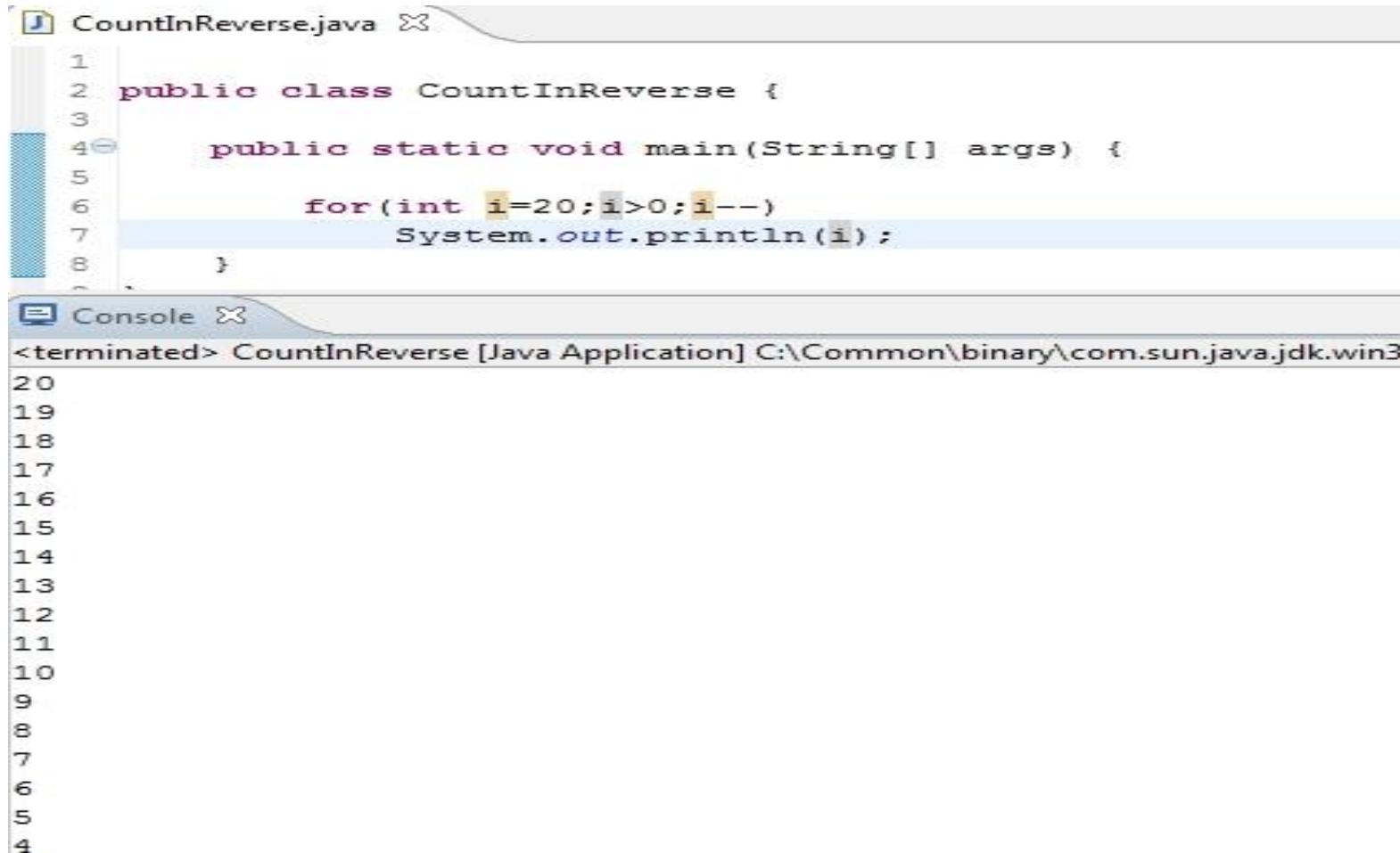
```
<terminated> CountInReverse [Java Application] C:\Common\binary\com.sun.java.jdk.win3
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
```

## For Loop

- Allows execution of same code a number of times
- `for(initializationexpression;loopcondition;stepexpression)`

```
{  
    statement1;  
    statement2;  
}
```

# For Loop



The screenshot shows a Java development environment with two tabs: "CountInReverse.java" and "Console".

**CountInReverse.java:**

```
1 public class CountInReverse {
2
3     public static void main(String[] args) {
4         for(int i=20;i>0;i--)
5             System.out.println(i);
6     }
7 }
```

**Console:**

```
<terminated> CountInReverse [Java Application] C:\Common\binary\com.sun.java.jdk.win32\jre\bin\javaw.exe
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
```

## Exercise

- Implement function that displays the table of any number n
- Implement exponent function that calculates the power of certain number (both power and base should be configurable)

# Solution for calculating tables

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class table {

    public static void main(String[] args) throws IOException {
        System.out.println("Please enter the number"+'\n');
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int s =Integer.parseInt(br.readLine());
        System.out.println('\n'+"Printing table for "+s );
        int i=1;
        for(i=1;i<21;i++)
            System.out.println(s+"*"+i+"="+s*i);
    }
}

```

Problems @ Javadoc Declaration

<terminated> table [Java Application] C:\Program

Please enter the number

13

Printing table for 13

13\*1=13  
 13\*2=26  
 13\*3=39  
 13\*4=52  
 13\*5=65  
 13\*6=78  
 13\*7=91  
 13\*8=104  
 13\*9=117  
 13\*10=130  
 13\*11=143  
 13\*12=156  
 13\*13=169  
 13\*14=182  
 13\*15=195  
 13\*16=208  
 13\*17=221  
 13\*18=234  
 13\*19=247  
 13\*20=260

# Solution for calculating exponent

```
import java.util.Scanner;

public class exponent {
    public static void main(String[] args) {
        System.out.println("Enter the base");
        Scanner sc = new Scanner(System.in);
        int base = Integer.parseInt(sc.nextLine());

        System.out.println("Enter the exponent");

        int exponent = Integer.parseInt(sc.nextLine());
        sc.close();
        int ans=1;
        for(int i=1;i<=exponent;i++)
            ans=ans*base;
        System.out.println("calculating");
        System.out.println(base+"^"+exponent+"="+ans);
    }
}
```

```
<terminated> exponent [Java Application] C:\F
Enter the base
8
Enter the exponent
3
calculating
8^3=512
```

# Java Collections

# What is a Collection?

- Group of objects
- These are used to store, retrieve, manipulate the data
- e.g. ledger -> a map to keep name of person and his accounts  
mailbox-> to keep all mails

# Java Collection Framework

- Java data structures to manipulate and represent collections
- Advantages
  - Reduces programming effort
  - If appropriate collection is used performance can be optimized
  - Gives API that can be reused

# Java Collections and Interfaces

| INTERFACES |         | IMPLEMENTATION |                |            |                                   |
|------------|---------|----------------|----------------|------------|-----------------------------------|
|            |         | Hashtable      | ResizableArray | Tree       | LinkedList<br>HasTable+LinkedList |
| SET        | Hashset |                |                | TreeSet    | LinkedHasSet                      |
| LIST       |         | ArrayList      |                | LinkedList |                                   |
| QUEUE      |         |                |                |            |                                   |
| MAP        | HashMap |                |                |            | LinkedHashMap                     |

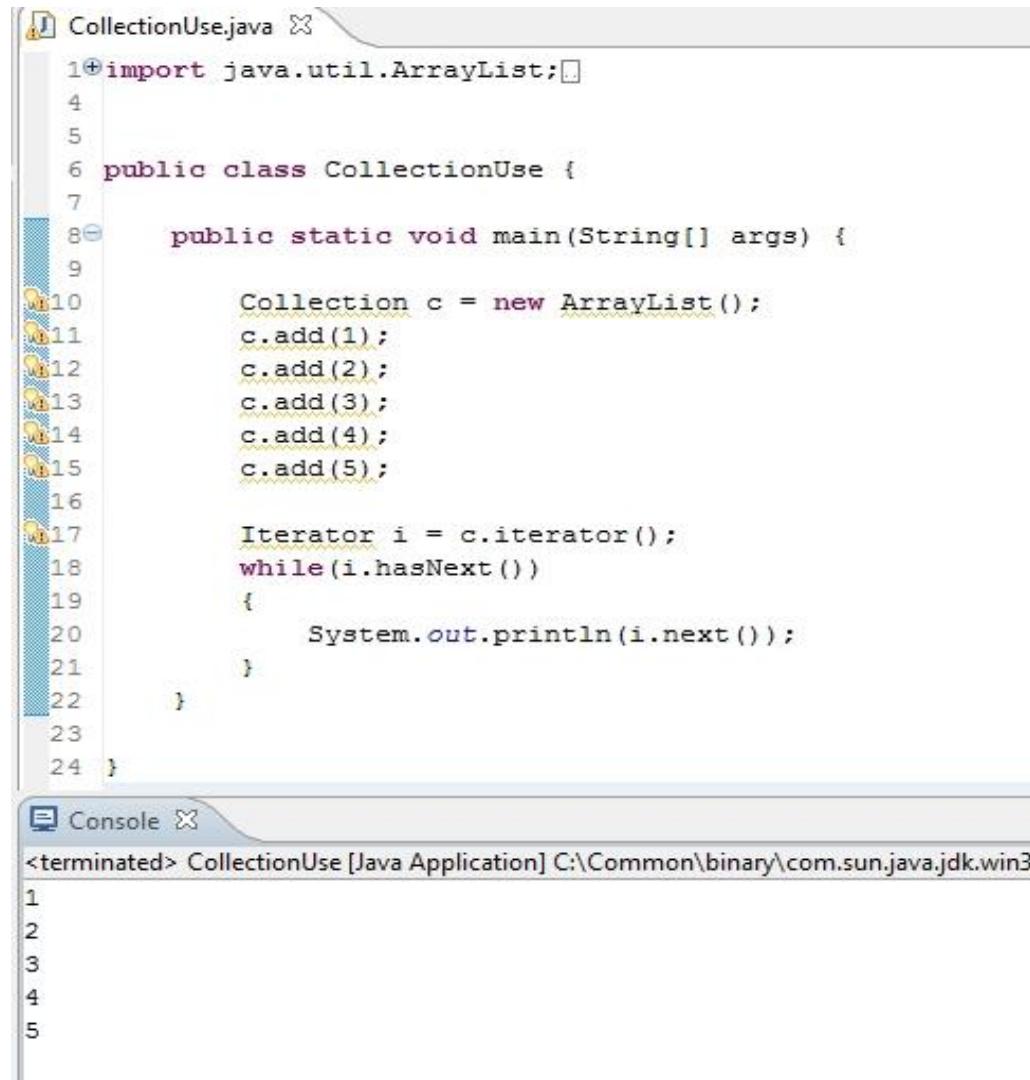
## Collection Interface

- The root interface in collection hierarchy
- Set, List, and Queue interfaces extend this interface
- This interface extends Iterable interface

# Methods in Collection interface

|                                          |                                                                                                                                                                                  |
|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| boolean                                  | <a href="#"><u>add(E e)</u></a> Ensures that this collection contains the specified element (optional operation).                                                                |
| boolean                                  | <a href="#"><u>addAll(Collection&lt;? extends E&gt; c)</u></a> Adds all of the elements in the specified collection to this collection (optional operation).                     |
| void                                     | <a href="#"><u>clear()</u></a> Removes all of the elements from this collection (optional operation).                                                                            |
| boolean                                  | <a href="#"><u>contains(Object o)</u></a> Returns true if this collection contains the specified element.                                                                        |
| boolean                                  | <a href="#"><u>containsAll(Collection&lt;?&gt; c)</u></a> Returns true if this collection contains all of the elements in the specified collection.                              |
| boolean                                  | <a href="#"><u>equals(Object o)</u></a> Compares the specified object with this collection for equality.                                                                         |
| int                                      | <a href="#"><u>hashCode()</u></a> Returns the hash code value for this collection.                                                                                               |
| boolean                                  | <a href="#"><u>isEmpty()</u></a> Returns true if this collection contains no elements.                                                                                           |
| <a href="#"><u>Iterator&lt;E&gt;</u></a> | <a href="#"><u>iterator()</u></a> Returns an iterator over the elements in this collection.                                                                                      |
| boolean                                  | <a href="#"><u>remove(Object o)</u></a> Removes a single instance of the specified element from this collection, if it is present (optional operation).                          |
| boolean                                  | <a href="#"><u>removeAll(Collection&lt;?&gt; c)</u></a> Removes all of this collection's elements that are also contained in the specified collection (optional operation).      |
| boolean                                  | <a href="#"><u>retainAll(Collection&lt;?&gt; c)</u></a> Retains only the elements in this collection that are contained in the specified collection (optional operation).        |
| int                                      | <a href="#"><u>size()</u></a> Returns the number of elements in this collection.                                                                                                 |
| <a href="#"><u>Object[]</u></a>          | <a href="#"><u>toArray()</u></a> Returns an array containing all of the elements in this collection.                                                                             |
| <T> T[]                                  | <a href="#"><u>toArray(T[] a)</u></a> Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array. |

# Collection



The screenshot shows an IDE interface with two tabs: 'CollectionUse.java' and 'Console'.

**CollectionUse.java:**

```
1+import java.util.ArrayList;
4
5
6 public class CollectionUse {
7
8     public static void main(String[] args) {
9
10         Collection c = new ArrayList();
11         c.add(1);
12         c.add(2);
13         c.add(3);
14         c.add(4);
15         c.add(5);
16
17         Iterator i = c.iterator();
18         while(i.hasNext())
19         {
20             System.out.println(i.next());
21         }
22     }
23
24 }
```

**Console:**

```
<terminated> CollectionUse [Java Application] C:\Common\binary\com.sun.java.jdk.win3.
1
2
3
4
5
```

## Methods to add and remove

- Remove(Object element)
  - Removes specified element from the collection, if it is present
  - Returns true if element is removed as a result of this call
- Add(E element)
  - Adds specified element to the collection
  - Returns true if element is added

# Traversing a collection object

- Iterator
  - Enables you to traverse through a collection and remove elements from collection selectively
- for-each
  - Allows to traverse a collection using for loop

# Traversal

```
import java.util.ArrayList;□

public class CollectionUse {

    public static void main(String[] args) {

        Collection<Integer> c = new ArrayList<Integer>();
        c.add(1);
        c.add(2);
        c.add(3);
        c.add(4);
        c.add(5);

        Iterator i = c.iterator();

        for(Integer j: c)
        {
            System.out.println(j);
        }
        while(i.hasNext())
        {
            System.out.println(i.next());
        }
    }
}
```

## Set Interface

- Represents a collection that can't contain duplicates
  - Roll number of students (Unique for each student)
  - Ids of employees

# Implementations of Set Interface

- HashSet
- TreeSet
- LinkedHashSet

## HashSet

- Faster than tree set, but unordered
- Constant time operations for (add, remove, contains and size)

## TreeSet

- Implements SortedSet
- Can be used if value ordered iteration is required

# Tree Set

```
CollectionUse.java X
1+import java.util.ArrayList;□
7
8
9 public class CollectionUse {
10
11    public static void main(String[] args) {
12
13        Set<Integer> ts = new TreeSet<Integer>();
14        ts.add(1);
15        ts.add(5);
16        ts.add(4);
17        ts.add(3);
18
19        for(Integer j : ts)
20            System.out.println(j);
21    }
22
23 }
24
```

```
Console X
<terminated> CollectionUse [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1
1
3
4
5
```

## LinkedHashSet

- Implemented as HashTable with ordering
- Provides insertion – ordered iteration(least recently added to most recently)

# LinkedHashSet

```
CollectionUse.java X
1+import java.util.ArrayList;
2
3
10 public class CollectionUse {
11
12     public static void main(String[] args) {
13
14         Set<Integer> ts = new LinkedHashSet<Integer>();
15         ts.add(1);
16         ts.add(5);
17         ts.add(4);
18         ts.add(3);
19         ts.add(5);
20
21         for(Integer j : ts)
22             System.out.println(j);
23     }
24
25 }
26
```

```
Console X
<terminated> CollectionUse [Java Application] C:\Common\binary\com.sun.java.jdk.win32.x86_1.6.0.013
1
5
4
3
```

## List Interface

- An ordered collection
- Can contain duplicate elements
- The list's objects can generally be accessed by the index in which they are added

# ArrayList

- Resizable-array implementation of the list interface
- Offers constant time positional access
- Each array list has a capacity
  - The capacity grows automatically as elements are added to it

## Map Interface

- Handles key/value pair
- A map can't contain duplicate keys

## SortedMap Interface

- Map maintains its mapping in ascending key order
- Sorted maps are used for naturally ordered collections of key/value pairs, such as dictionaries and telephone dictionaries

# Implementations of Map Interface

- **LinkedHashMap**
  - Used when insertion-order is required
- **TreeMap**
  - Implements SortedMap Interface
  - Used for key ordered collection-view iteration
- **HashMap**
  - Can be used when faster accessed is required without ordering

## Exercise

- Which data structure should you use to maintain phone directory
- Which data structure would you use to implement the names of queue of people who have entered the bank and should be served according to First come first serve basis
- Subway wants to maintain list of all customer names who have visited and had sandwich. Which data structure is suitable for this?

# Coding Guidelines for Package Names

- Package names are written in all lower case to avoid conflict with the names of classes or interfaces.
- Companies use their reversed Internet domain name to begin their package names—for example, com.example.mypackage for a package named mypackage created by a programmer at example.com.
- Name collisions that occur within a single company need to be handled by convention within that company, perhaps by including the region or the project name after the company name (for example, com.example.region.mypackage).
- Packages in the Java language itself begin with java. or javax.

---

---

# Thank you!

**Pristine**

702, Raaj Chambers, Old Nagardas Road, Andheri (E), Mumbai-400 069. INDIA

[www.edupristine.com](http://www.edupristine.com)

Ph. +91 22 3215 6191