**PROJECT REPORT**

**On**

**“THE PING PONG GAME”**

Using

Core Java 1.7 and JRE7

In partial fulfillment of the prescribed course for

The Award of

## BACHELOR OF TECHNOLOGY IN

## COMPUTER SCIENCE ENGINEERING (CSE)

****

**GURU PREMSUKH MEMORIAL COLLEGE OF ENGINEERING**

245, Budhpur Village, G. T. Karnal Road, Delhi-110036  
Ph: 011-27207048, 49

**Submitted By:**

GOURAV GOYAL

CSE(5thSem)

06913102711

**CONTENTS**

1. Acknowledgement
2. Certificate
3. Preface
4. Introduction
5. Purpose
6. System Analysis
7. Hardware and Software specification
8. SRS
9. Introduction
   1. Purpose
   2. Scope
   3. Defination,Acronym
   4. Overview
10. Overall description

2.1 Product Aspective

3. Specific requirement

3.1 External interface

3.2 Functional requirement

4. System design

5. System testing

6. Flow chart

1. Coding
2. Snapshots
3. Conclusion
4. Bibliography

ACKNOWLEDGMENT

With due respect, we express our deep sense of gratitude to our respected and learned guides and mentors **Mr. PradeepKamboj**(HOD, computer science and engineering Department, GPMCE **)** and**Dr.Narendra Kumar** (Director, GPMCE) for their co-operation and guidance. We acknowledge their invaluable help and suggestions in implementing the project. We are thankful to them for sparing their valuable time and encouraging us at every stage.

We are also grateful to faculty of the COMPUTER science Department, GPMCE for their ready support.

Submitted By:-

GOURAV GOYAL (06913102711)

CERTIFICATE

This is to certify that this major project entitled as “**THE PING PONG GAME**” has been completed by **GOURAV GOYAL (069),** during fifth semester of Bachelor of Technology in computer science engineering from Guru Premsukh Memorial College of Engineering. The matter embodied in this project has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Date: 01 October 2013

Mr. Pradeep Kamboj Dr. Narendra Kumar

(HOD, Cse Department, GPMCE) (Director, GPMCE)

PREFACE

The Project is based is basically a “The PING PONG GAME”. It includes all necessary details including the address entries of an individual.

It uses java technology & JFrame for its implementation. It includes window based skills. Under all these, I developed the PING PONG GAME.

INTRODUCTION

Pong (marketed as PONG) is one of the earliest arcade video games; it is a tennis sports game featuring simple two-dimensional graphics.

The game was originally manufactured by Atari Incorporated (Atari), who released it in 1972. Allan Alcorn created Pong as a training exercise assigned to him by Atari co-founder Nolan Bushnell. Bushnell based the idea on an electronic ping-pong game included in the Magnavox Odyssey, which later resulted in a lawsuit against AtariPong is a two-dimensional sports game that simulates table tennis. The player controls an in-game paddle by moving it vertically across the left side of the screen, and can compete against either a computer controlled opponent or another player controlling a second paddle on the opposing side. Players use the paddles to hit a ball back and forth.

The aim is for a player to earn more points than the opponent; points are earned when one fails to return the ball to the other.

PURPOSE

The project is about to handle all the the possibility of playing the simple “Ping Pong game”. Also it manages resources which were managed and handled by manpower previously. The main purpose of the project is to integrate the learning aspects of applets and java scripting to the real world environment gamming and to do in consistent manner so that complex functions can be handled smoothly by any technical or non-technical persons.

The project aims at the following matters:

• The simple user friendly game.

• Uses the move function of pad.

• To change the color and speed of ball to increaseDifficulty and excitement.

• Consistently update the score.

SYSTEM ANALYSIS

Existing System

System Analysis is a detailed study of the various operations performed by a system and their relationships within and outside of the system. Here the key question is- what all problems exist in the present system? What must be done to solve the problem? Analysis begins when a user or manager begins a study of the program using existing system.

During analysis, data collected on the various files, decision points and transactions handled by the present system. The commonly used tools in the system are Data Flow Diagram, interviews, etc. Training, experience and common sense are required for collection of relevant information needed to develop the system. The success of the system depends largely on how clearly the problem is defined, thoroughly investigated and properly carried out through the choice of solution. A good analysis model should provide not only the mechanisms of problem understanding but also the frame work of the solution. Thus it should be studied thoroughly by collecting data about the system. Then the proposed system should be analyzed thoroughly in accordancewiththeneeds.

Benefit analysis

In the current system we need to keep a number of records related to the student and want to enter the details of the student and the marks manually. In this system only the teacher or the school authority views the mark of the student and they want to enter the details of the student. This is time consuming.

Proposed system

In our proposed system we have the provision for adding the details of the students by themselves. So the overhead of the school authorities and the teachers is become less. Another advantage of the system is that it is very easy to edit the details of the student and delete a student when it found unnecessary. The marks of the student are added in the database and so students can also view the marks whenever they want.

Feasibility study

Whatever we think need not be feasible .It is wise to think about the feasibility of any problem we undertake. Feasibility is the study of impact, which happens in the organization by the development of a system. The impact can be either positive or negative. When the positives nominate the negatives, then the system is considered feasible. Here the feasibility study can be performed in two ways such as technical feasibility and Economical Feasibility.

We can strongly say that it is technically feasible, since there will not be much difficulty in getting required resources for the development and maintaining the system as well. All the resources needed for the development of the software as well as the maintenance of the same is available in the organization here we are utilizing the resources which are available already.

Hardware and Software specifications

**Hardware and Software Requirements :**

Before starting any new language we must know what are its hardware and software requirements. Also, adding to our additional knowledge, we were taught about the installation process. So this section tells us about the Hardware and Software requirements and installation of Core JAVA.

**Hardware Requirements:**

* 512mb RAM

**Software Requiremenrs:**

* JDK 1.6

**Operating System :**

* Microsoft Windows XP/ Vista/ Windows7/ Windows8

**Phases in the Project:**

* System Requirements and Analysis
* Design Phase
* Coding
* Testing

**SRS(Software Requirements Specification) OF PING-PONG**

**GAME**

**1. Introduction**

* 1. **Purpose**

The Software Requirements Specification (SRS) will provide a detailed description of the requirements of the PING PONG game. This SRS will allow for a complete understanding of what is to be expected of the game. The clear understanding of the game and its functionality will allow for the correct software to be developed for the end user and will be used for the development of the future stages of the project. This SRS will provide the foundation for the project. From this SRS, the Game can be designed, constructed, and finally tested.

This SRS will be used by the software engineers constructing the Notepad and the end users. The software engineers will use the SRS to fully understand the expectations of this Notepad to construct the appropriate software. The end users will be able to use this SRS as a “test” to see if the software engineers will be constructing the system to their expectations. If it is not to their expectations the end users can specify how it is not to their liking and the software engineers will change the SRS to fit the end users’ needs.

**1.2 Scope** :

The software product to be produced is a Basic indoor version of the very popular table-tennis game which can be played by one person.

The main objective of this game is to create the highest score although the existence of various levels and correspondingly increase the difficulties which makes the gamer to play tough.

**1.3Definitions, Acronyms and Abbreviations** :

SRS – Software Requirement Specification

Subjective Satisfaction – The overall satisfaction of the system

End users – The people who will be actually using the system

**1.4 Overview :**

The SRS is organized into two main sections. The first is the overall description and the second is the Specific Requirements. The overall description will describe the requirements of the HMS from a general high level perspective. The Specific Requirements section will describe in detail the requirements of the system.

1. **The Overall Description :**

It describes the general factors that affect the product and its requirements.This section does not state specific requirements but it provides a background for those specific requirements and makes them easier to understand.

**2.1Product Perspective :**

The is an independent stand–alone system. It is totally self contained.

**2.1.1 Hardware Interfaces**

The HMS will be placed on PC’s throughout the hostel.

**2.1.2 Software Interfaces**

The whole PING-PONG Game is made using JAVA using jdk1.6.

**2.1.3 Communication Interfaces**

None

**2.1.4 Memory Constraints**

512 MB which is required by the jdk.

**2.1.5 Site Adaptation Requirement**.

The terminals at client site will have to support the hardware and software interfaces specified in above sections.

**2.1.6 Game Controls :**

* Use of mouse
* Movement of paddle via mouse
* Change colors of the ball
* Exiting the application
* Increasing the speed
* Raising the difficulty at each level
* Increase the score
* Reset the score

**2.4Assumptions and Dependencies**

None.

**2.5 Apportioning of requirements**

Not required.

1. **Specific Requirements**

This section contains all the software requirements at a level of detail, that when combined with the system context diagram, use cases, and use case descriptions, is sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements.

* 1. **External Interfaces**

The Ping-Pong Game will use the standard input/ output devices for a personal computer. These include the following :

* Keyboard
* Mouse
* Monitor
* Printer(not mandatory)
  1. **Functional Requirements**

Functional Requirements define the fundamental actions that system must perform. The functional requirements for the system are divided into three main categories, student information system, room information system and billing section. For further details, refer to use cases.

1. **File Management System**
   1. The system shall input file name
   2. The system shall input the extension of the file
   3. The system shall ask the user to specify the location to save the file
   4. The system shall ask the user which file to open and from what location
   5. The system shall shut down the notepad screen if the user asks for it

**2. Editing**

2.1 The system shall copy the selected text

2.2 The system shall cut the selected text

2.3 The system shall paste the selected text

2.4 The system shall select the whole document when specified by the user

**3. Setting**

3.1 The system shall change the font of the text to the one specified by the user.

**4. Formatting**

4.1 The system shall implement line wrapping when specified by the user.

**SYSTEM DESIGN**

**Design of Game:**

This ping pong Game uses the movement of the mouse to control the paddle.

Moving the mouse to the left will set the paddle to the left side and Moving the mouse to the right will set the paddle to the right side.

Ball should hit the paddle in order to increase the score.

As the time passes speed of the ball start increases and also the color of the ball.

When a player miss the ball his score reset to the zero and ball sets the color to Blue.

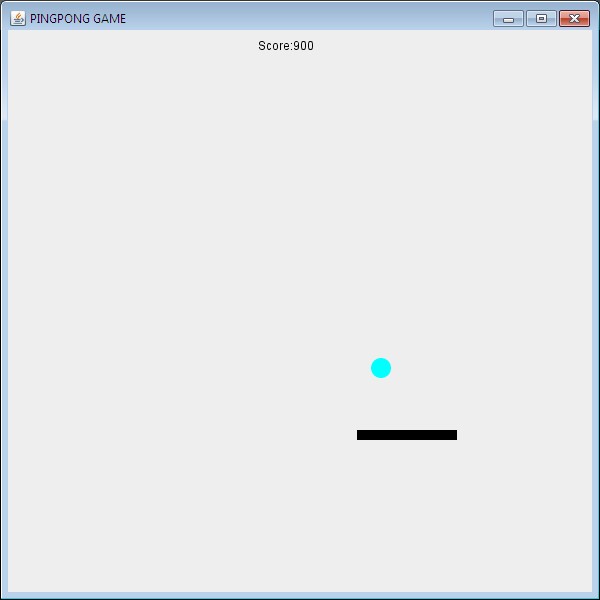
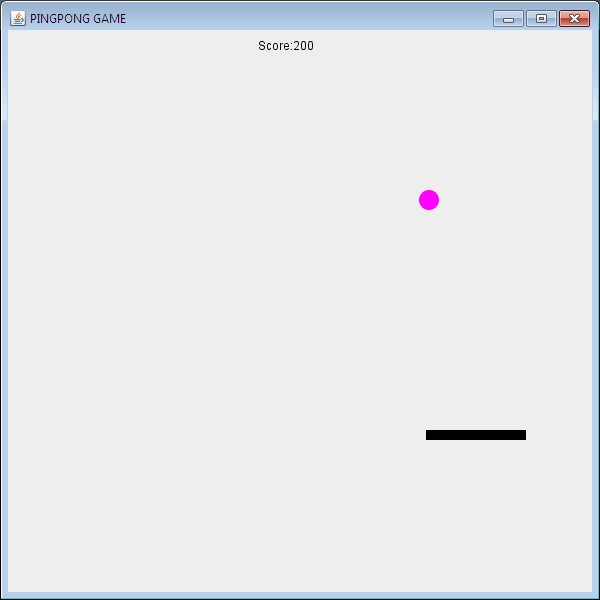
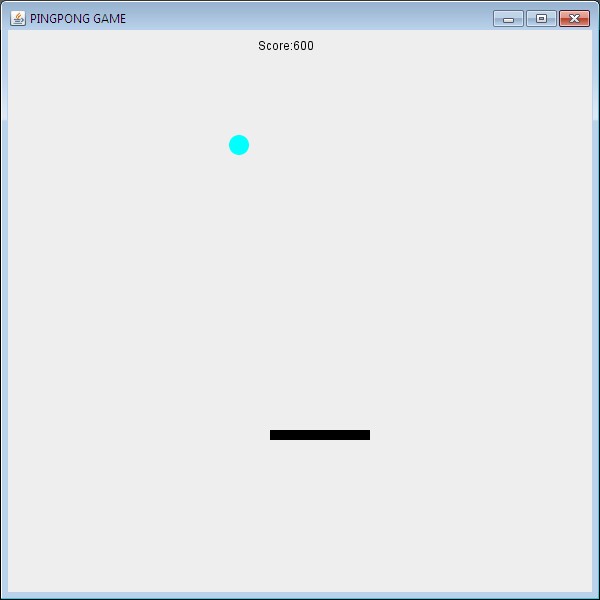


Figure-: Design of Ping-Pong Game

**SYSTEM TESTING**





**SYSTEM DEVELOPMENT**

**CODING**

**Packages Imported :**

* import java.awt.\*;
* importjava.awt.event.\*;
* import java.io.\*;
* importjavax.swing.\*;
* importjava.util.\*;
* importjava.awt.PrintJob.\*;

**Built-in Classes Used :**

* Font
* JMenuBar
* JMenuItem
* Frame
* Choice
* Button
* JTextArea
* FileReader
* FileWriter
* LineNumberReader
* String Reader
* String
* FileReader
* FileWriter

**User Defined Classes :**

* Font
* Ss
* Mynotepad
* myfr

**Interfaces Implemented:**

* ActionListener
* Mouse Listener

CODING

/\*Code for position and movement of Ball

import java.awt.geom.Ellipse2D;

class Ball extends Ellipse2D.Float{

publicintx\_speed, y\_speed;

privateint d;

public Ball(int diameter)

{

//RANDOM LOCATION OF NEW BALL

super( (int)(Math.random() \* (Canvas.width - 20) + 1),0, diameter, diameter );

this.d = diameter;

this.x\_speed = (int)(Math.random() \* 5 + 5);

this.y\_speed = (int)(Math.random() \* 5 + 5);

}

public void move()

{

//TO STRIKE BACK FROM THE CORNERS

if (super.x< 0 || super.x>Canvas.width - d)

x\_speed = -x\_speed;

if (super.y< 0 || super.y>Canvas.height - d)

y\_speed = -y\_speed;

super.x += x\_speed;

super.y += y\_speed;

}

}

/\*CODE FOR JFRAME SIZE

importjavax.swing.JFrame;

public class Canvas extends JFrame{

//WIDTH & HEIGHT OF FRAME

staticint width = 600;

staticint height = 600;

privateThreadAnimation ta;

privatePaintSurfaceps;

public Canvas(){

setTitle("PINGPONG GAME");

setSize(width,height);

ps = new PaintSurface();

super.add(ps);

ta = new ThreadAnimation(this);

ta.start();

setVisible(true);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public static void main(String args[]){

new Canvas();

}

}

/\*CODE FOR INTERACT BALL WITH PADDLE

importjava.awt.Color;

importjava.awt.Graphics;

import java.awt.Graphics2D;

importjava.awt.RenderingHints;

importjava.awt.Shape;

importjava.awt.event.MouseEvent;

importjava.awt.event.MouseMotionAdapter;

import java.awt.geom.Rectangle2D;

importjavax.swing.JComponent;

public class PaintSurface extends JComponent {

//INITIAL POSITION OF PADDLE

intpaddle\_x = 0;

intpaddle\_y = 400;

// INITIAL SCORE

int score = 0;

floatenglish = 1.0f;

Ball ball;

// COLORS OF THE BALL

Color[] color = {Color.BLUE, Color.ORANGE,

Color.MAGENTA, Color.GREEN,

Color.CYAN, Color.RED};

intcolorIndex;

publicPaintSurface(){

addMouseMotionListener(new MouseMotionAdapter()

{

public void mouseMoved(MouseEvent e)

{

if (e.getX() - 30 - paddle\_x> 5)

english = 1.5f;

else if (e.getX() - 30 - paddle\_x< -5)

english = -1.5f;

else

english = 1.0f;

paddle\_x = e.getX() - 30;

}

} );

ball = new Ball(20);

}

public void paint(Graphics g){

Graphics2D g2 = (Graphics2D)g;

g2.setRenderingHint(

RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

Shape paddle = new Rectangle2D.Float(paddle\_x, paddle\_y, 100, 10);

//PADLE SIZE

g2.setColor(color[colorIndex % 6]);

// IF PADDLE HITS THE BALL

if (ball.intersects(paddle\_x, paddle\_y, 100, 10) &&ball.y\_speed> 0)

{

ball.y\_speed = -ball.y\_speed;

ball.x\_speed = (int)(ball.x\_speed \* english);

//COLOR CHANGE

if (english != 1.0f)

{

colorIndex++;

}

// INCREASING SCORE

score +=100;

}

// IF PADDLE MISS THE BALL

if (ball.getY() + ball.getHeight() >= Canvas.height)

{

ball = new Ball(20);

score = 0;

colorIndex = 0;

}

ball.move();

g2.fill(ball);

g2.setColor(Color.BLACK); //COLOR OF PADDLE n SCORE

g2.fill(paddle);

g2.drawString("Score:" + score, 250, 20);

}

}

/\*CODING FOR THREAD ANIMATION

importjavax.swing.JFrame;

public class ThreadAnimation extends Thread {

JFrame f;

ThreadAnimation(JFrame f){

this.f = f;

}

public void run() {

while(true){

f.repaint();

try {

Thread.sleep(20);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**Why Java?**

**The Java Programming Language**

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

* Simple
* Object oriented
* Robust
* Secure
* Architecture neutral
* Portable
* High performance
* Multithreaded
* Dynamic

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called ***Java bytecodes* —the platform-independent codes interpreted by the interpreter on the Java platform.** The interpreter parses and runs each Java bytecode instruction on the computer. Compilation happens just once, interpretation occurs each time the program is executed.

You can think of Java bytecodes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM.

Java bytecodes help make "write once, run anywhere" possible. You can compile your program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows7, a Solaris workstation, or on an iMac.

**The Java Platform**

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows7, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

* The *Java Virtual Machine* (Java VM)
* The *Java Application Programming Interface* (Java API)

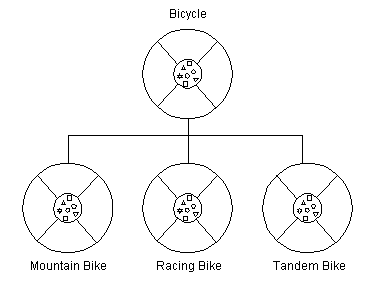
The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces, these libraries are known as *packages*.

**Native code** is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring performance close to that of native code without threatening portability. The Java compiler allows you to cascade references to class and instance methods and variables together, resulting in constructs like the one that appears in the sample program.

## *Inheritance*

Generally speaking, objects are defined in terms of classes. You know a lot about an object by knowing its class. Even if you don't know what a penny-farthing is, if I told you it was a bicycle, you would know that it had two wheels, handle bars, and pedals.

Object-oriented systems take this a step further and allow classes to be defined in terms of other classes. For example, mountain bikes, racing bikes, and tandems are all kinds of bicycles. In object-oriented terminology, mountain bikes, racing bikes, and tandems are all [**subclasses**](javascript:var%20meth=openWin;%20meth('subclass');) of the bicycle class. Similarly, the bicycle class is the [**superclass**](javascript:var%20meth=openWin;%20meth('superclass');)of mountain bikes, racing bikes, and tandems. This relationship is shown in the following figure ..

.

**Figure-4 : Object Oriented Relationship**

Each subclass [**inherits**](javascript:var%20meth=openWin;%20meth('inheritance');) state (in the form of variable declarations) from the super class. Mountain bikes, racing bikes, and tandems share some states: cadence, speed, and the like. Also, each subclass inherits methods from the super class. Mountain bikes, racing bikes, and tandems share some behaviours: braking and changing pedalling speed, for example.

However, subclasses are not limited to the state and behaviors provided to them by their superclass. Subclasses can add variables and methods to the ones they inherit from the superclass. Tandem bicycles have two seats and two sets of handle bars; some mountain bikes have an extra set of gears with a lower gear ratio.

Subclasses can also [**override**](javascript:var%20meth=openWin;%20meth('overriding');) inherited methods and provide specialized implementations for those methods. For example, if you had a mountain bike with an extra set of gears, you would override the "change gears" method so that the rider could use those new gears.

You are not limited to just one layer of inheritance. The inheritance tree, or class [**hierarchy**](javascript:var%20meth=openWin;%20meth('hierarchy');)**,** can be as deep as needed. Methods and variables are inherited down through the levels. In general, the farther down in the hierarchy a class appears, the more specialized its behavior.

The Object class is at the top of class hierarchy, and each class is its descendant (directly or indirectly). A variable of type Object can hold a reference to any object, such as an instance of a class or an array. Object provides behaviors that are required of all objects running in the Java Virtual Machine. For example, all classes inherit Objects toString method, which returns a string representation of the object.

Inheritance offers the following benefits:

* Subclasses provide specialized behaviours from the basis of common elements provided by the superclass. Through the use of inheritance, programmers can reuse the code in the superclass many times.
* Programmers can implement superclasses called [**abstract classes**](javascript:var%20meth=openWin;%20meth('abstract%20class');)that define "generic" behaviours. The abstract superclass defines and may partially implement the behaviour, but much of the class is undefined and unimplemented. Other programmers fill in the details with specialized subclasses.

## *Interface*

In English, an interface is a device or a system that unrelated entities use to interact. According to this definition, a remote control is an interface between you and a television set, the English language is an interface between two people, and the protocol of behaviour enforced in the military is the interface between people of different ranks. Within the Java programming language, an [interface](javascript:var%20meth=openWin;%20meth('interface');) is a device that unrelated objects use to interact with each other. An interface is probably most analogous to a protocol (an agreed on behaviour). In fact, other object-oriented languages have the functionality of interfaces, but they call their interfaces protocols.

The bicycle class and its class hierarchy defines what a bicycle can and cannot do in terms of its "bicycleness." But bicycles interact with the world on other terms. For example, a bicycle in a store could be managed by an inventory program. An inventory program doesn't care what class of items it manages as long as each item provides certain information, such as price and tracking number. Instead of forcing class relationships on otherwise unrelated items, the inventory program sets up a protocol of communication. This protocol comes in the form of a set of constant and method definitions contained within an interface. The inventory interface would define, but not implement, methods that set and get the retail price, assign a tracking number, and so on.

To work in the inventory program, the bicycle class must agree to this protocol by implementing the interface. When a class implements an interface, the class agrees to implement all the methods defined in the interface. Thus, the bicycle class would provide the implementations for the methods that set and get retail price, assign a tracking number, and so on.

* You use an interface to define a protocol of behavior that can be implemented by any class anywhere in the class hierarchy. Interfaces are useful for the following:
* Capturing similarities among unrelated classes without artificially forcing a class relationship.
* Declaring methods that one or more classes are expected to implement.
* Revealing an object's programming interface without revealing its class.

## *Object*

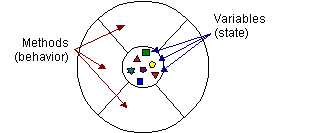
Objects are key to understanding [**object-oriented**](javascript:var%20meth=openWin;%20meth('object-oriented%20design');) technology. You can look around you now and see many examples of real-world objects: your dog, your desk, your television set, your bicycle.

These real-world objects share two characteristics: They all have state and behaviour. For example, dogs have state (name, color, breed, hungry) and behavior (barking, fetching, and wagging tail). Bicycles have state (current gear, current pedal cadence, two wheels, number of gears) and behavior (braking, accelerating, slowing down, changing gears). Software objects are modeled after real-world objects in that they too have state and behavior. A software object maintains its state in one or more [**variables**](javascript:var%20meth=openWin;%20meth('variable');)**.** A variable is an item of data named by an identifier. A software object implements its behavior with [**methods**](javascript:var%20meth=openWin;%20meth('method');). A method is a function (subroutine) associated with an object.

**Definition of Object :** An object is a software bundle of variables and related methods.

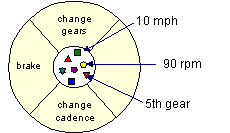
You can represent real-world objects by using software objects. You might want to represent real-world dogs as software objects in an animation program or a real-world bicycle as software object in the program that controls an electronic exercise bike. You can also use software objects to model abstract concepts. For example, an event is a common object used in GUI window systems to represent the action of a user pressing a mouse button or a key on the keyboard.

The following illustration is a common visual representation of a software object:



**Figure-5: Visual Representation of software object**

Everything that the software object knows (state) and can do (behaviour) is expressed by the variables and the methods within that object. A software object that modelled your real-world bicycle would have variables that indicated the bicycle's current state: its speed is 10 mph, its pedal cadence is 90 rpm, and its current gear is the 5th gear. These variables are formally known as [**instance variables**](javascript:var%20meth=openWin;%20meth('instance%20variable');)because they contain the state for a particular bicycle object, and in object-oriented terminology, a particular object is called an **instance.**



**Figure -6: Bicycle modeled as a software object**

In addition to its variables, the software bicycle would also have methods to brake, change the pedal cadence, and change gears. (The bike would not have a method for changing the speed of the bicycle, as the bike's speed is just a side effect of what gear it's in, how fast the rider is pedalling, whether the brakes are on, and how steep the hill is.) These methods are formally known as [**instance methods**](javascript:var%20meth=openWin;%20meth('instance%20method');) because they inspect or change the state of a particular bicycle instance.

The object diagrams show that the object's variables make up the center, or nucleus, of the object. Methods surround and hide the object's nucleus from other objects in the program. Packaging an object's variables within the protective custody of its methods is called [**encapsulation**](javascript:var%20meth=openWin;%20meth('encapsulation');) this conceptual picture of an object-a nucleus of variables packaged within a protective membrane of methods-is an ideal representation of an object and is the ideal that designers of object-oriented systems strive for. However, it's not the whole story. Often, for practical reasons, an object may wish to expose some of its variables or hide some of its methods. In the Java programming language, an object can specify one of four access levels for each of its variables and methods. The access level determines which other objects and classes can access that variable or method. Variable and method access in Java is covered in Controlling Access to Members of a Class.

Encapsulating related variables and methods into a neat software bundle is a simple yet powerful idea that provides two primary benefits to software developers:

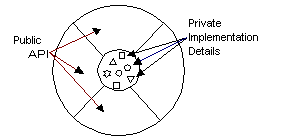
* **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects. Also, an object can be easily passed around in the system. You can give your bicycle to someone else, and it will still work.
* **Information hiding:** An object has a public interface that other objects can use to communicate with it. The object can maintain private information and methods that can be changed at any time without affecting the other objects that depend on it. You don't need to understand the gear mechanism on your bike to use it.

## *Class*

In the real world, you often have many objects of the same kind. For example, your bicycle is just one of many bicycles in the world. Using object-oriented terminology, we say that your bicycle object is an [**instance**](javascript:var%20meth=openWin;%20meth('instance');) of the class of objects known as bicycles. Bicycles have some state (current gear, current cadence, two wheels) and behaviour (change gears, brake) in common. However, each bicycle's state is independent of and can be different from that of other bicycles. When building bicycles, manufacturers take advantage of the fact that bicycles share characteristics, building many bicycles from the same blueprint. It would be very inefficient to produce a new blueprint for every individual bicycle manufactured.

In object-oriented software, it's also possible to have many objects of the same kind that share characteristics: rectangles, employee records, video clips, and so on. Like the bicycle manufacturers, you can take advantage of the fact that objects of the same kind are similar and you can create a blueprint for those objects. A software blueprint for objects is called a [**class**](javascript:var%20meth=openWin;%20meth('class');)**.**

**Definition of Class :** A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.

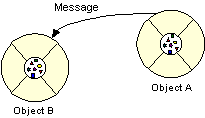


**Fig -7: Class Implementation**

## *Message*

A single object alone is generally not very useful. Instead, an object usually appears as a component of a larger program or application that contains many other objects. Through the interaction of these objects, programmers achieve higher-order functionality and more complex behaviour. Your bicycle hanging from a hook in the garage is just a bunch of titanium alloy and rubber; by itself, the bicycle is incapable of any activity. The bicycle is useful only when another object (you) interacts with it (pedal).

Software objects interact and communicate with each other by sending messages to each other. When object A wants object B to perform one of B's methods, object A sends a message to object B

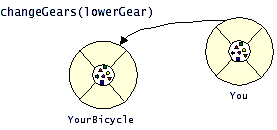


**Fig 8: Sending of message between objects**

Sometimes, the receiving object needs more information so that it knows exactly what to do; for example, when you want to change gears on your bicycle, you have to indicate which gear you want. This information is passed along with the message as parameters.

The next figure shows the three components that comprise a message:

1. The object to which the message is addressed (Your Bicycle)
   * `The name of the method to perform (change Gears)
2. Any parameters needed by the method (lower Gear)



**Fig 9: Components of a message**

These three components are enough information for the receiving object to perform the desired method. No other information or context is required.

Messages provide two important benefits :

* An object's behaviour is expressed through its methods, so (aside from direct variable access) message passing supports all possible interactions between objects.

**STEPS in GAME**

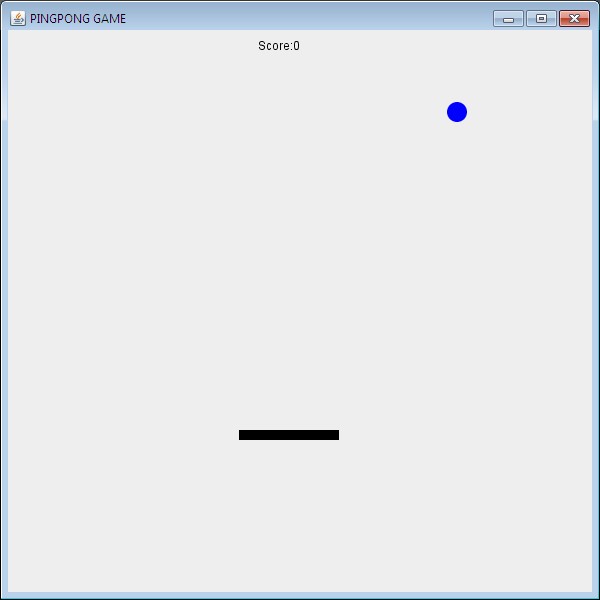
**GAME: The Ping-Pong Game**

This game is divided into various modules. Various modules are:

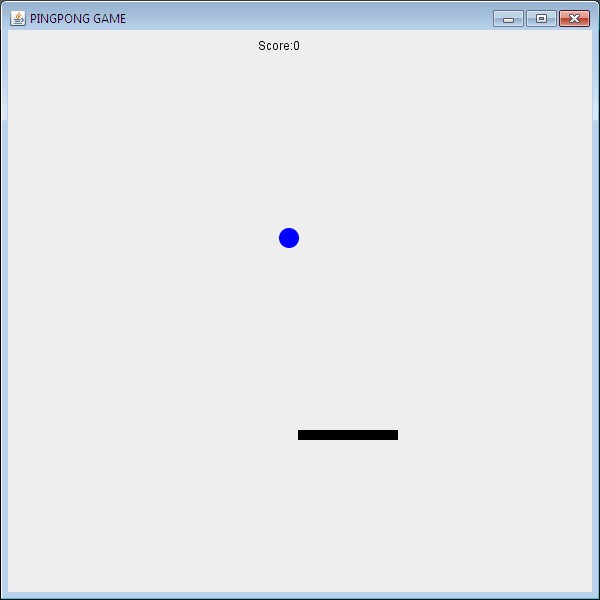
* Firstly I made the made the canvas by extending JFrame
* Then after the above module I give the size of the canvas. This size is adjustable by mouse.
* I also give the title to the Canvas.
* Then I made a class of ball and set various properties of the ball. Like diameter and shape of the ball.
* I set the postion and speed of the ball (x axis and y axis) and function for bouncing back from the walls.
* After all this another module which uses mouseListener interface is made. This module was the most important and time-taking module. In this module I define all the events which will take place on actionPerformed.
* In main module I set the various color of the ball and the speed which increase after each level.
* I set the default score and paddle movement and various interaction of ball with the paddle.

**SNAPSHOTS**

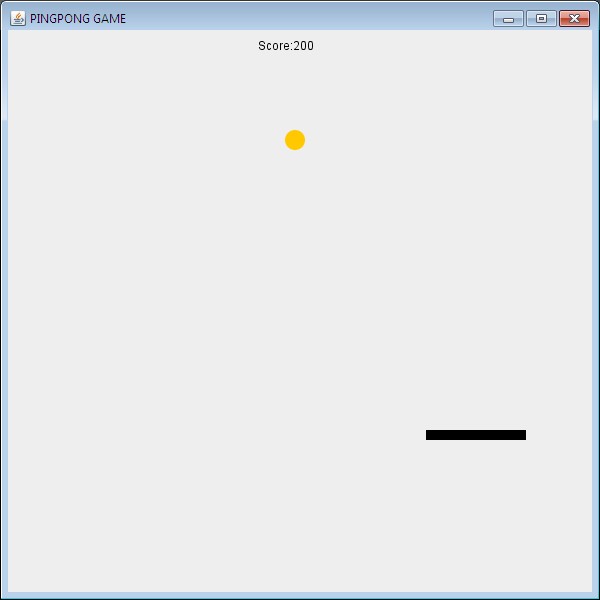
INITIAL SCORE:



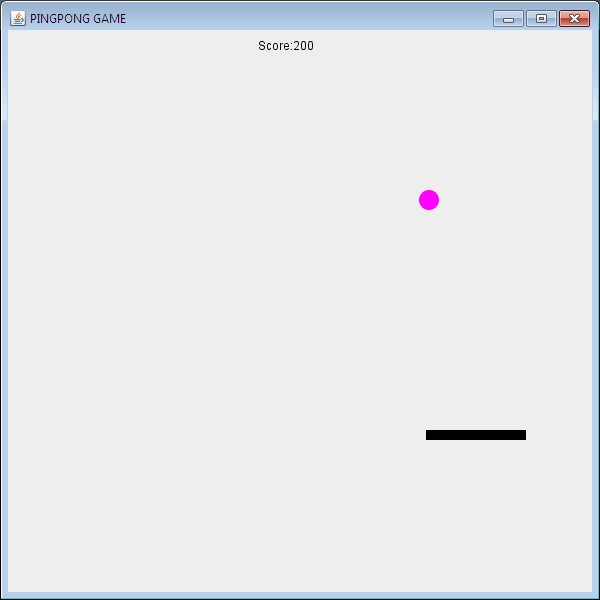
INITIAL BALL = BLUE AND LEVEL 1:



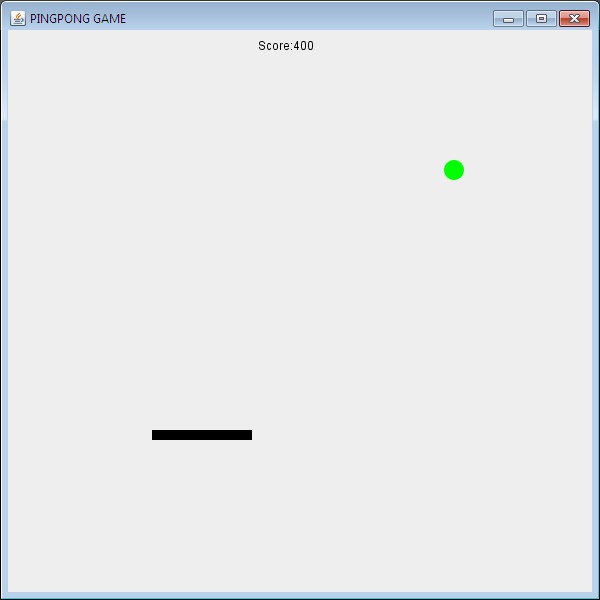
LEVEL 2: BALL = YELLOW



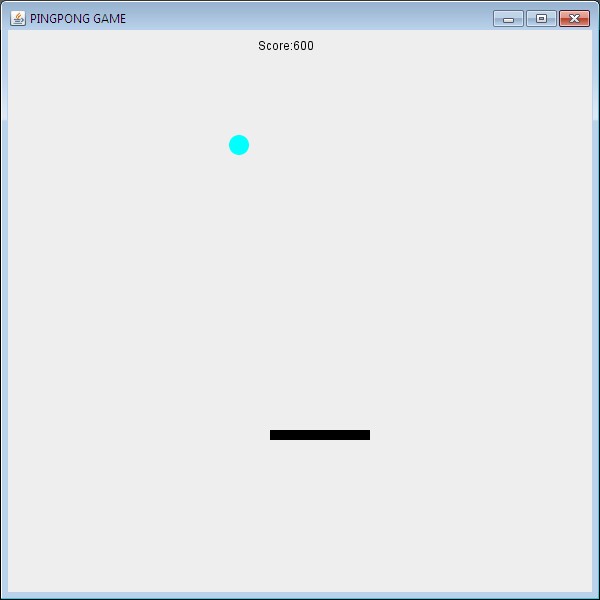
LEVEL 3: BALL = MAGENTA



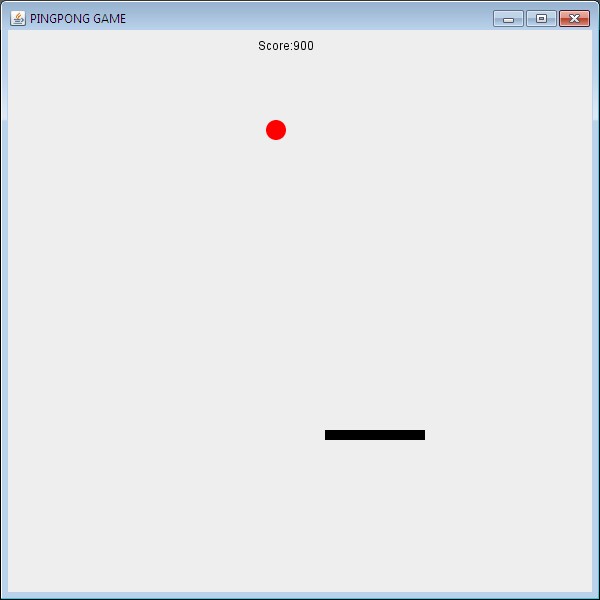
LEVEL 4: BALL = GREEN



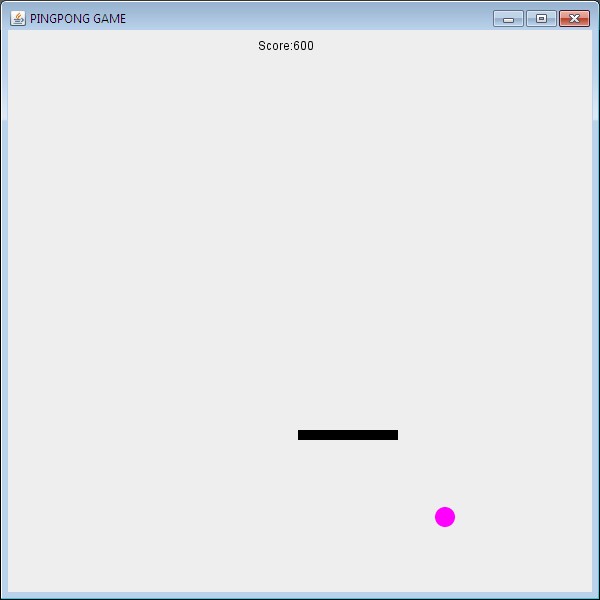
LEVEL 5: BALL = CYAN



LEVEL 6: BALL = RED



PADDLE MISSES THE BALL



**CONCLUSION**

Our project is only a humble venture to satisfy the needs in an Institution. Several user friendly coding have also adopted. This package shall prove to be a powerful package in satisfying all the requirements of the organization.  
The objective of software planning is to provide a frame work that enables the manger to make reasonable estimates made within a limited time frame at the beginning of the software project and should be updated regularly as the project progresses.

**FUTURE WORK AND SCOPE**

Below are the points which I would like to augment in my current text editor :

* I Would like to make this game Multiplayer so friends can play and challenge each other .
* I would like to add more features, levels to make it more exciting and interesting.
* It would be able to play online across the internet by using the features of servelets.
* It would be supported by various devices like android,apple phones and desktops

**BIBLIOGRAPHY**

1. JAVA COMPLETE REFERENCE BY HERBEDT SCHILDT – 7th edition.
2. Head First Java 2nd edition by Kathy Sierra & Bert Bates .Gurudeb.
3. ThinkingInJava by bruceeckel – 4th edition.
4. Java All-In-One Desk Reference for Dummies by douglowe. – 2nd edition.
5. WORLD WIDE WEB