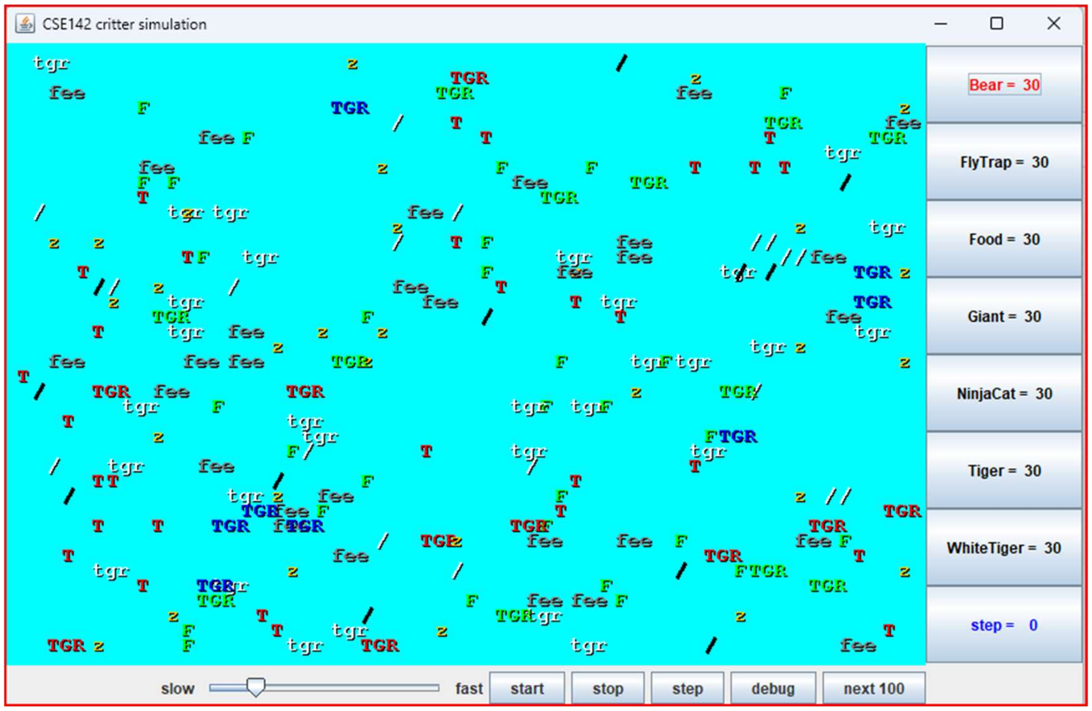
Assignment 3

NARESH A/L M NARENDRAN (UNU2200290)

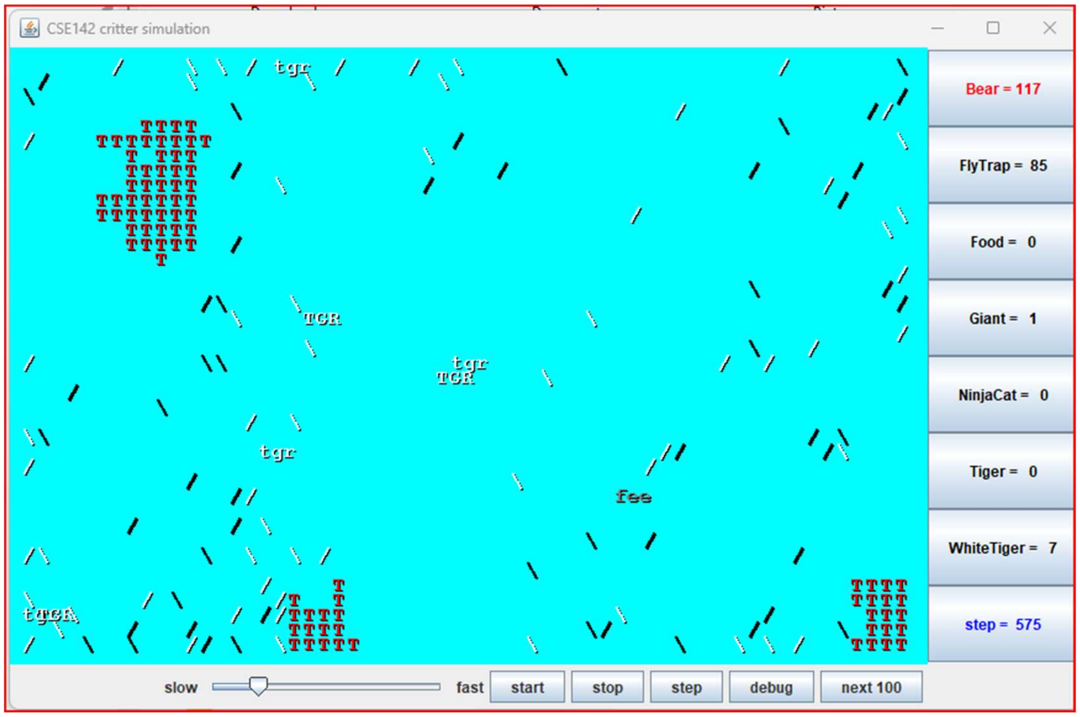
ITWM5013 SOFTWARE DESIGN AND DEVELOPMENT

TABLE OF CONTENTS

FINAL PROJECT – ANIMAL KINGOM



At start up



After running.

1.0 ABSTRACT

Object-oriented programming (OOP) is a programming paradigm that structures software design around data, or objects, as opposed to logic and functions. The definition of an object is a data field with unique attributes and behavior. Instead of focusing on the logic required to manipulate objects, OOP emphasizes the objects that developers wish to manipulate. This programming methodology is well-suited for large, complex, and actively updated or maintained programs. Abstraction, encapsulation, inheritance, and polymorphism are among these features. In Object-Oriented Programming, inheritance is a fundamental concept (OOP). Using inheritance, we can accurately model a problem and reduce the number of lines we must write. A data field with unique attributes and behavior can be defined as an object. Instead of focusing on the logic required to manipulate objects, OOP emphasizes the objects that developers wish to manipulate. This programming methodology is well-suited for large, complex, and actively updated or maintained programs. This includes manufacturing and design applications as well as mobile applications. Writing this report and developing this application in Java using Object-Oriented Programming (OOP) can be an onerous task requiring knowledge of several large libraries and advanced Java OOP concepts. In this assignment, you are required to write a set of classes that define the behavior of specific animals, and you will be provided with a program that simulates a world populated by numerous free-roaming animals. You are defining the differences in behavior between species of animals, which will vary. To produce the result, intensive research and testing of errors were conducted.

2.0 INTRODUCTION

This capstone assignment provides the opportunity to practice with a set of classes that define the characteristics of specific animals. The output of the program depicts the numerous animals roaming the area.

This project concentrates on the use of multiple objects and classes and how they interact. Students are given a large amount of code for this project, which includes a simulation of numerous animals roaming the environment. Distinct species of animals will exhibit distinct behaviors, and we are defining these distinctions.

Each class in the Java program that simulates the Animal Kingdom has a particular function. Here is a quick rundown of the classes concerned:

Critter.java

The foundation class for all creatures in the Animal Kingdom is represented by this class. It outlines typical characteristics and behaviors that other animal classes inherit, such as size, color, and modes of locomotion.

CritterFrame.java

This class, Critter.Frame.java, represents the program's graphical user interface (GUI) frame. It offers the window through which the Animal Kingdom simulation is shown, showing the animals and their interactions.

CritterInfo.java

This class represents an interface that offers details on an animal's attributes, such as its location, movement, and level of energy.

CritterMain.java

The main method to launch the simulation of the Animal Kingdom is contained in this class, which also serves as the program's entrance point.

CritterModel.java

This class acts as the simulation's model for Animal Kingdom. It records the positions, motions, and interactions of all the creatures in the simulation in addition to their states and behaviors.

CritterPanel.java

This class represents the GUI panel where the illustrations of the creatures are drawn and shown. During the simulation, it oversees rendering the animals and updating their positions.

FlyTrap.java

This class, FlyTrap.java, represents a particular species of animal in the Animal Kingdom. It defines its distinctive characteristics and behaviors, such as its capacity to catch other animals, by inheriting from the Critter class.

Food.java

In the Animal Kingdom simulation, this class depicts the food items that animals can eat to provide energy. It defines characteristics and actions associated with food, including its location and accessibility.

Giant.java

The Animal Kingdom's Giant species is represented by this class. It describes its characteristics and behaviors, such as its size and gait pattern, by inheriting from the Critter class.

NinjaCat.java

The NinjaCat is a particular species of animal that is represented by this class. It defines its distinctive characteristics and actions, such as its covert movement and contact with other creatures, by inheriting from the Critter class.

Tiger.java

This class represents the tiger, a particular species of animal in the animal kingdom. It derives from the Critter class and specifies its unique characteristics, like speed and hunting tendencies.

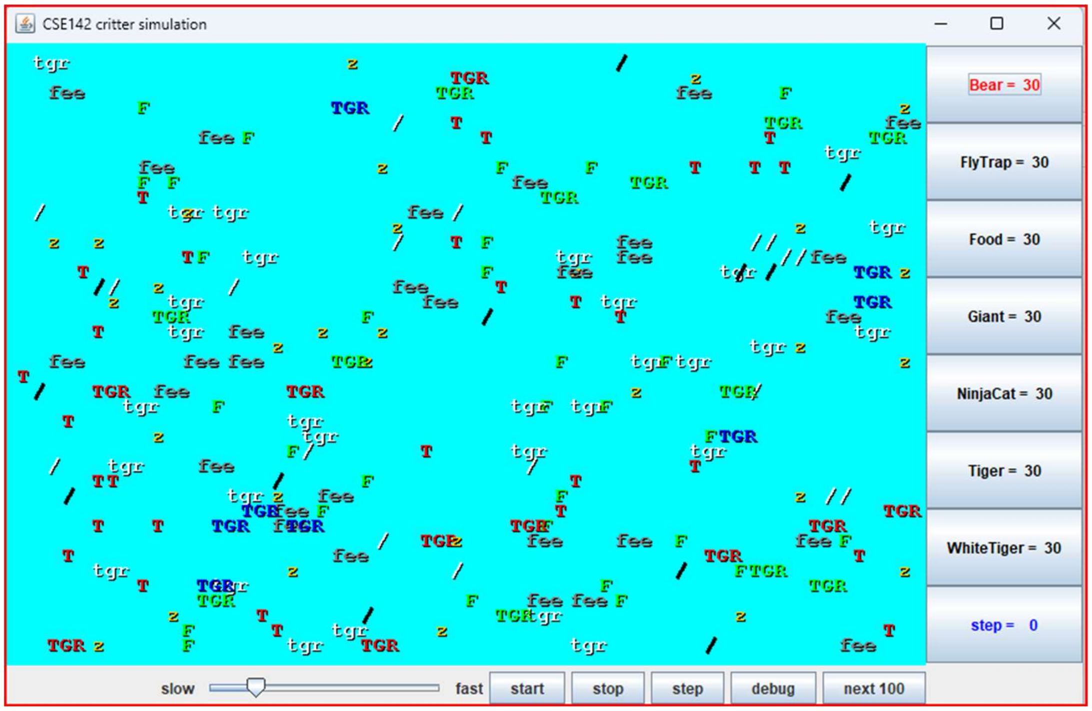
WhiteTiger.java

This class depicts the WhiteTiger, a particular species of animal in the Animal Kingdom. It derives from the Tiger class and establishes its distinctive characteristics, including its color and interactions with other animals.

Bear.Java

This class depicts the Bear, a particular species of animal in the Animal Kingdom. It derives from the Bear class and establishes its distinctive characteristics, including its color and interactions with other animals.

For this assignment, you will be provided with a substantial amount of code that executes the simulation. During operation, the simulation will resemble the image below:



3.0 DESCRIPTION

This project requires us to write Java code where Critter is the superclass with defined default behavior. We will be writing five classes, each representing a distinct type of Animal: Bear, Tiger, WhiteTiger, Giant, and NinjaCat. All classes that are correct should be subclasses of Critter. On each round of the simulation, each animal is asked three questions, such as "How should it behave?" and "What hue is it?" What string represents this animal? These three items of information are provided by three methods present in each Critter class, and it is the responsibility of the programmer to override these methods and program the appropriate behavior.

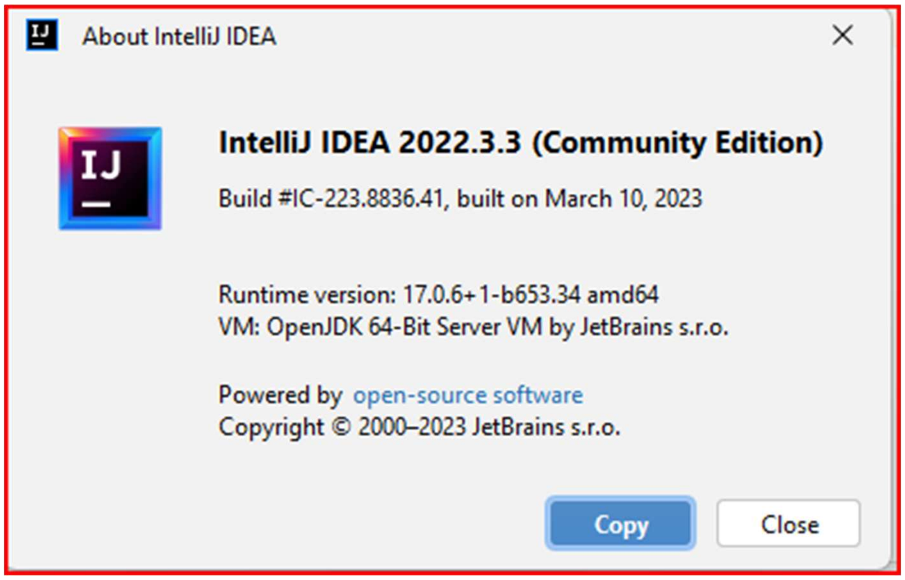
The code for these behaviors will be contained within each of the Critter classes, and I must program the behavior by overriding each of the methods within each class. The simulator determines the Critter's hue. I must designate a color using the color dot and color enter codes. Each option should have an equal chance of being chosen at random. So, I can utilize either an object or math.random(). The next method to be addressed is toString. This returns a string of text that instructs the simulator how to display your Critter. getMove is the concluding method that responds to the behaviors.

This procedure can only return a maximum of four items. action.Right, action.Left, action.Up. and action.Infect. To determine which direction the method should return, investigate the Critters' surroundings and environments with additional techniques. In all the Critters, there may come a time when one of the variables must be modified due to the Critter's multiple actions. It would be beneficial to add a move counter to getMove so that other methods can be used when necessary. As I've completed each class, I've confirmed that everything is functioning properly.

This is where testing is useful. I commented out the frame in the simulator to isolate the Critter I was working on the frame.add lines to the Critter primary file. This will allow me to isolate the Critter and verify my code's functionality step by step. For direction instructions, the simulator features a diagnostic mode that displays your Critter as an arrow pointing in the direction it is facing.

4.0 DESIGN

Animal Kingdom Game must be developed using the JAVA programming language. I use IntelliJ IDEA 2022.1.1 to accomplish my objectives (Community Edition). Classes are used to construct the complete structure of the project. It is necessary to compile with the JAVA 18 SDK and execute CritterMain.java as the main program to invoke the interface's main form.

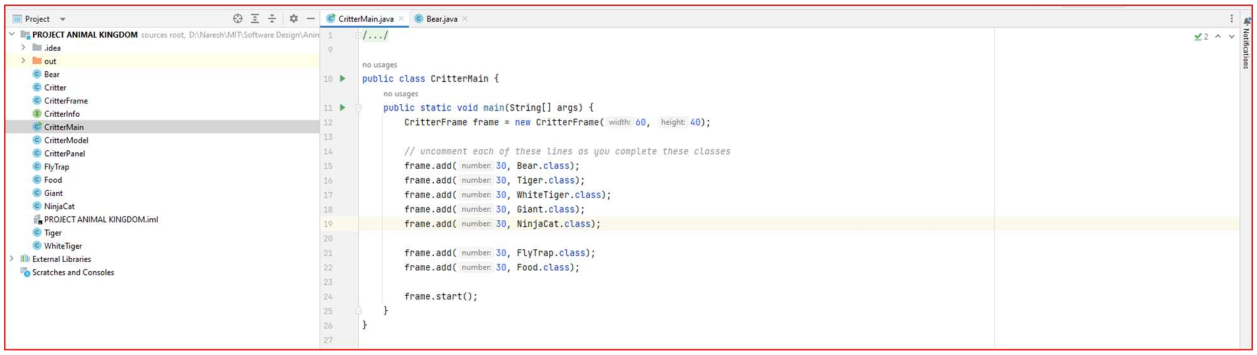


About IntelliJ IDEA

IntelliJ IDEA 2022.3.3 (Community Edition)
Build #IC-223.8836.41, built on March 10, 2023

Runtime version: 17.0.6+1-b653.34 amd64
VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

Powered by open-source software
Copyright © 2000–2023 JetBrains s.r.o.



```
PROJECT ANIMAL KINGDOM  sources root, D:\Naresh\MIT\Software Design\Anin
  .idea
  out
  Bear
  Critter
  CritterFrame
  CritterInfo
  CritterMain
  CritterModel
  CritterPanel
  FlyTrap
  Food
  Giant
  NinjaCat
  PROJECT ANIMAL KINGDOM.iml
  Tiger
  WhiteTiger
External Libraries
Scratches and Consoles
```

```java
/.../

public class CritterMain {
    public static void main(String[] args) {
        CritterFrame frame = new CritterFrame( width: 60,  height: 40);

        // uncomment each of these lines as you complete these classes
        frame.add( number: 30, Bear.class);
        frame.add( number: 30, Tiger.class);
        frame.add( number: 30, WhiteTiger.class);
        frame.add( number: 30, Giant.class);
        frame.add( number: 30, NinjaCat.class);

        frame.add( number: 30, FlyTrap.class);
        frame.add( number: 30, Food.class);

        frame.start();
    }
}
```

The above are the components of the 8 classes for the Animal Kingdom Project.

To accurately represent the traits and actions of animals, the Java program that represents the Animal Kingdom makes use of several fundamental Java principles. Let's talk about the applications of each of these ideas:

Classes

In Java, classes are used to specify an object's building blocks. Classes like Giant, NinjaCat, Tiger, and WhiteTiger are used in the Animal Kingdom application to represent many species of animals. Each class defines the characteristics (such as size, color, and speed) of a particular animal species or activity. (e.g., movement, interaction).

Objects

During program execution, objects—which are instances of classes—are produced. For each animal species or activity in the Animal Kingdom software, items are made to depict the animals with their distinctive traits and habits. As an illustration, items belonging to the Tiger or WhiteTiger classes would represent tigers in the Animal Kingdom, complete with their own traits and behaviors.

Inheritance

Java's idea of inheritance enables classes to take on characteristics and behaviors from a parent class. In the Animal Kingdom program, the hierarchical relationships between various animal species or behaviors are represented by inheritance. Classes like Tiger and WhiteTiger, for instance, may have their own distinct properties and behaviors in addition to inheriting common attributes and behaviors from a parent class like Critter.

Polymorphism

Java's idea of polymorphism enables objects of various classes to be viewed as belonging to a single parent class. Polymorphism enables the modeling of various animal species or behaviors in the Animal Kingdom application using a single interface or parent class. To handle various animal kinds in the application uniformly, objects of the Tiger, WhiteTiger, and Giant classes, for instance, can all be regarded as objects of the Critter class.

Encapsulation

Encapsulation is a Java concept that entails concealing a class's internal workings and only making the relevant data accessible through methods. Encapsulation is used in the Animal Kingdom software to group an animal's characteristics and actions into classes that can be accessed and modified with control. This promotes the preservation of data integrity and the abstraction of animal behavior.

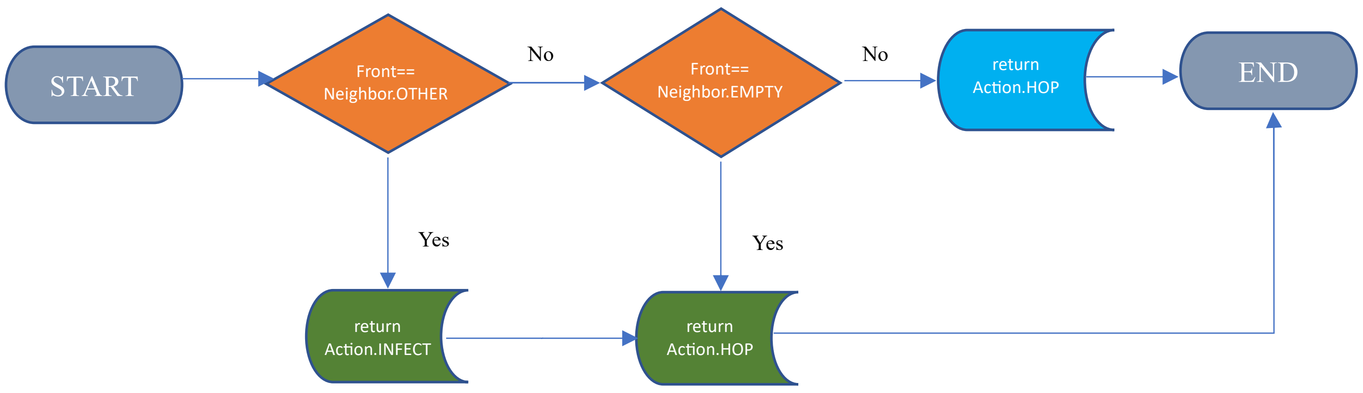
The Animal Kingdom program can represent the traits and actions of animals in a structured and organized way by making appropriate use of these Java ideas. Inheritance creates hierarchical relationships, polymorphism enables unified handling of various animal types, encapsulation ensures data integrity, and objects represent individual animals with their distinctive attributes and behaviors. All these factors work together to represent different animal species or behaviors. Collectively, these Java principles support the program's accurate representation of the animal kingdom and allow for the simulation of a range of animal behaviors and traits.

5.0 WORKFLOW AND LOGIC

There are 5 animal classes. I've used on of the main constructors getMove from 5 animal classes to demonstrate the flowchart sequence as following:

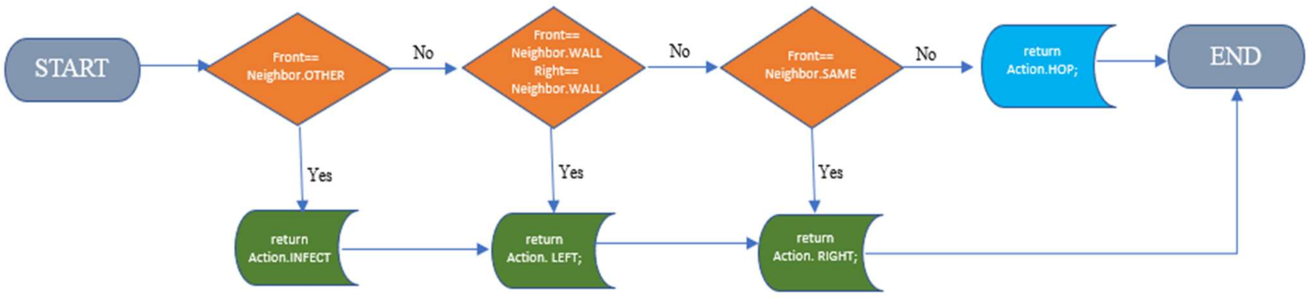
Bear

getMove()



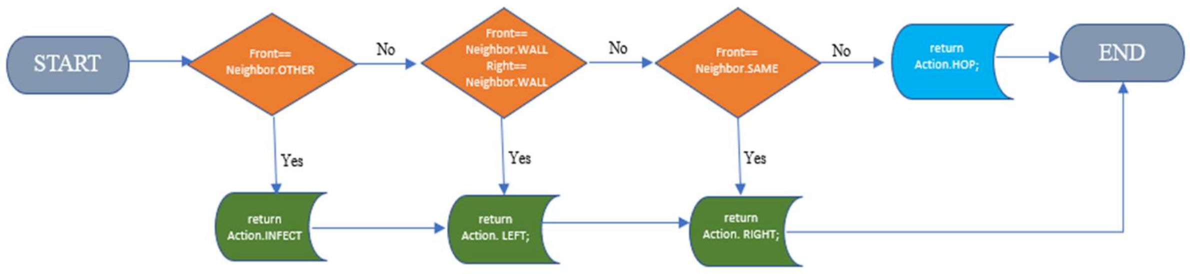
Tiger

getMove()



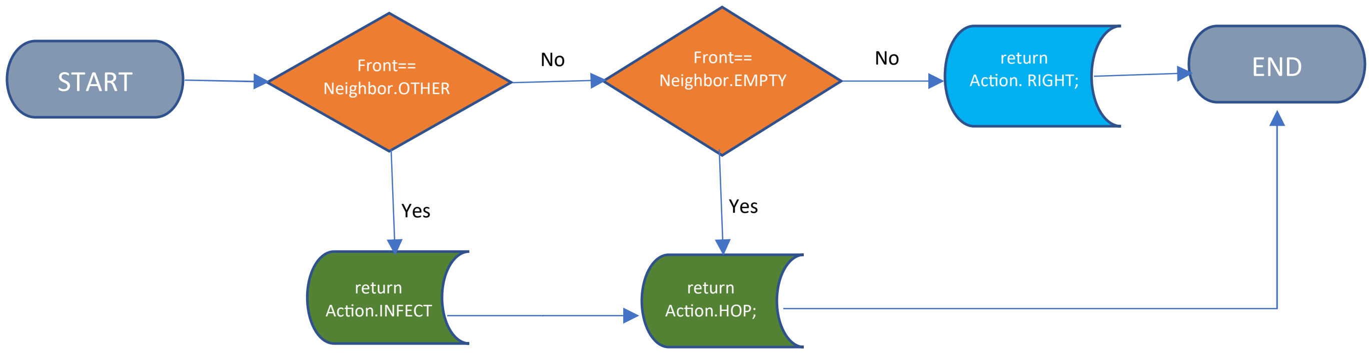
WhiteTiger

getMove()



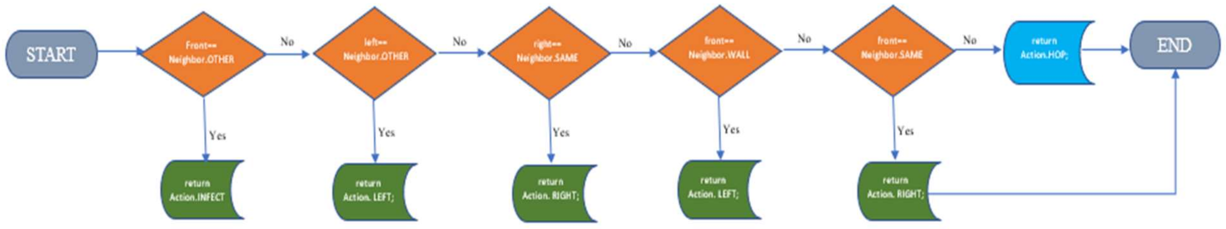
Giant

getMove()



NinjaCat

getMove()

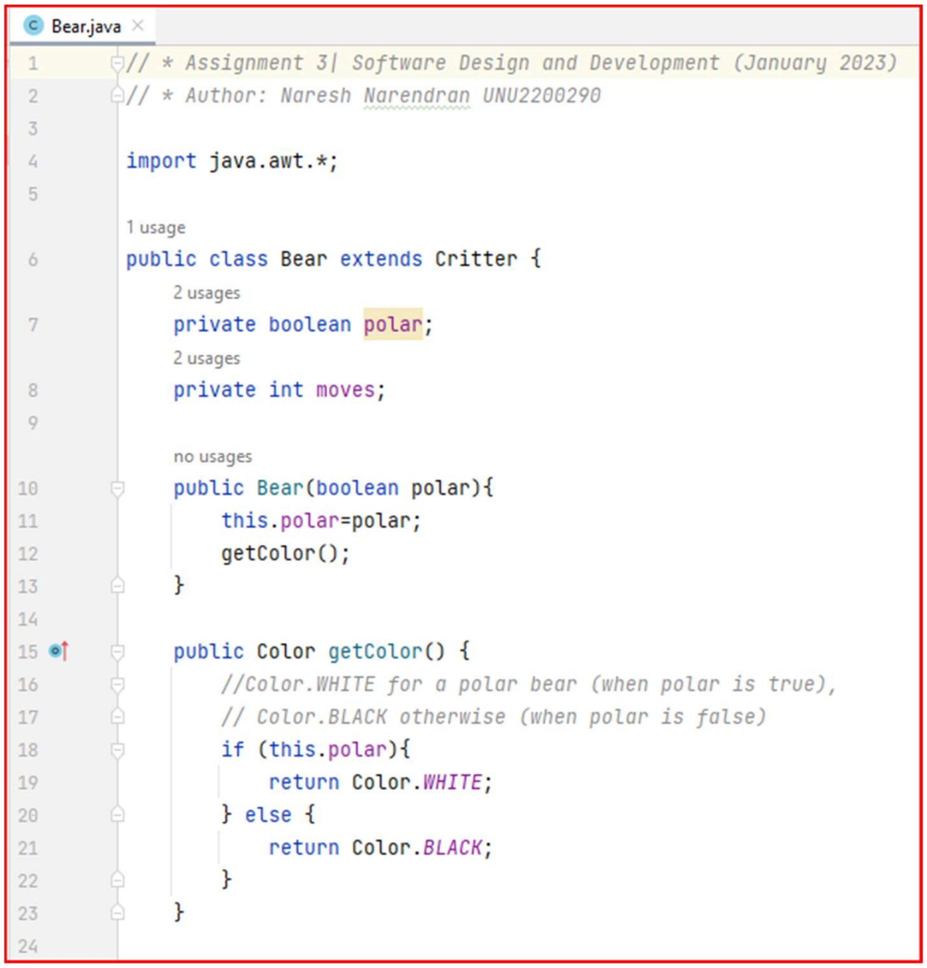


6.0 DEVELOPMENT

A total of 5 classes were given to build the source code.

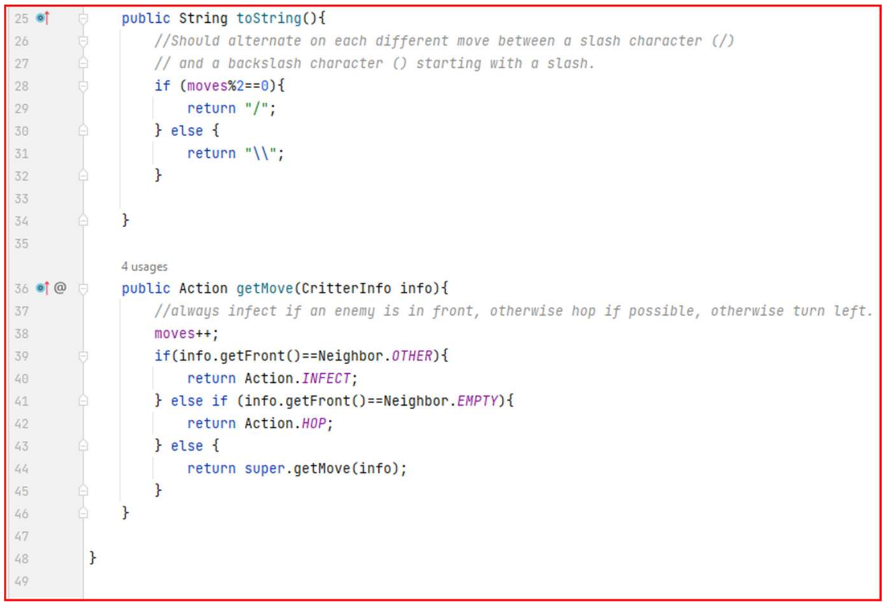
Bear.java

Constructor	**public Bear (boolean polar)**
getColor	Color.WHITE for a polar bear (when polar is true), Color.BLACK otherwise (when polar is false).
toString	Should alternate on each different move between a slash character (/) and a backslash character () starting with a slash.
getMove	always infect if an enemy is in front, otherwise hop, if possible, otherwise turn left.



Bear.java

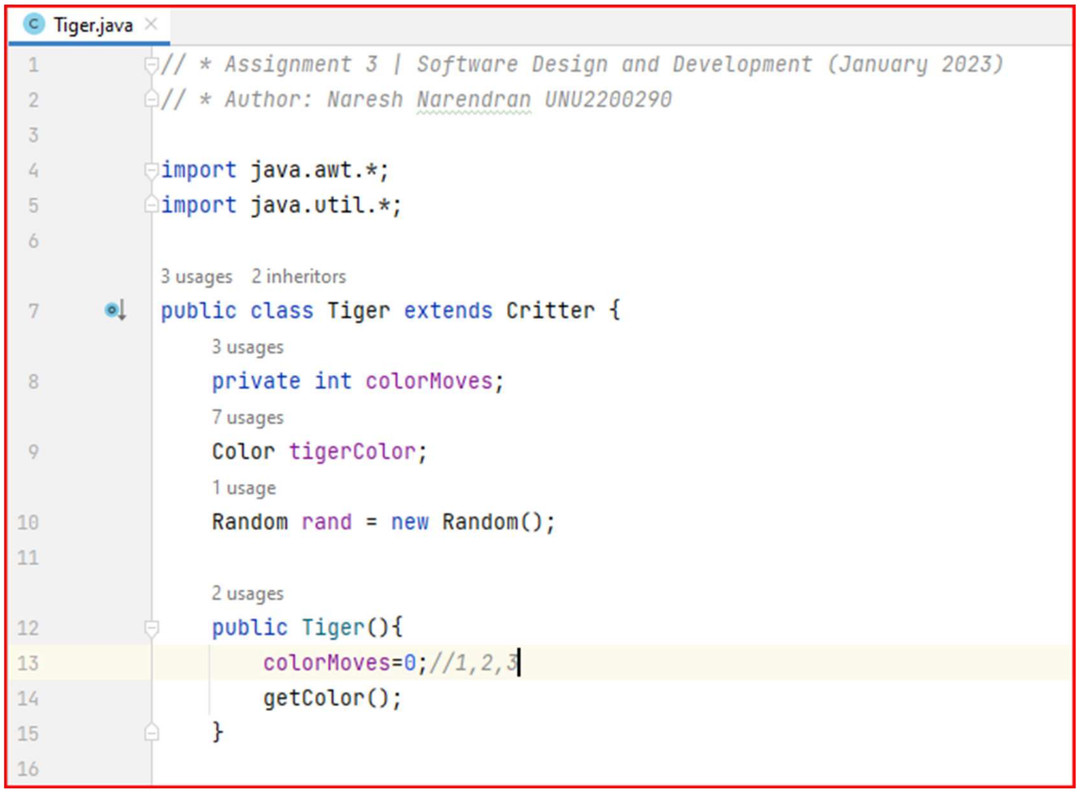
```
// * Assignment 3| Software Design and Development (January 2023)
// * Author: Naresh Narendran UNU2200290

import java.awt.*;

1 usage
public class Bear extends Critter {
    2 usages
    private boolean polar;
    2 usages
    private int moves;

    no usages
    public Bear(boolean polar){
        this.polar=polar;
        getColor();
    }

    public Color getColor() {
        //Color.WHITE for a polar bear (when polar is true),
        // Color.BLACK otherwise (when polar is false)
        if (this.polar){
            return Color.WHITE;
        } else {
            return Color.BLACK;
        }
    }

```



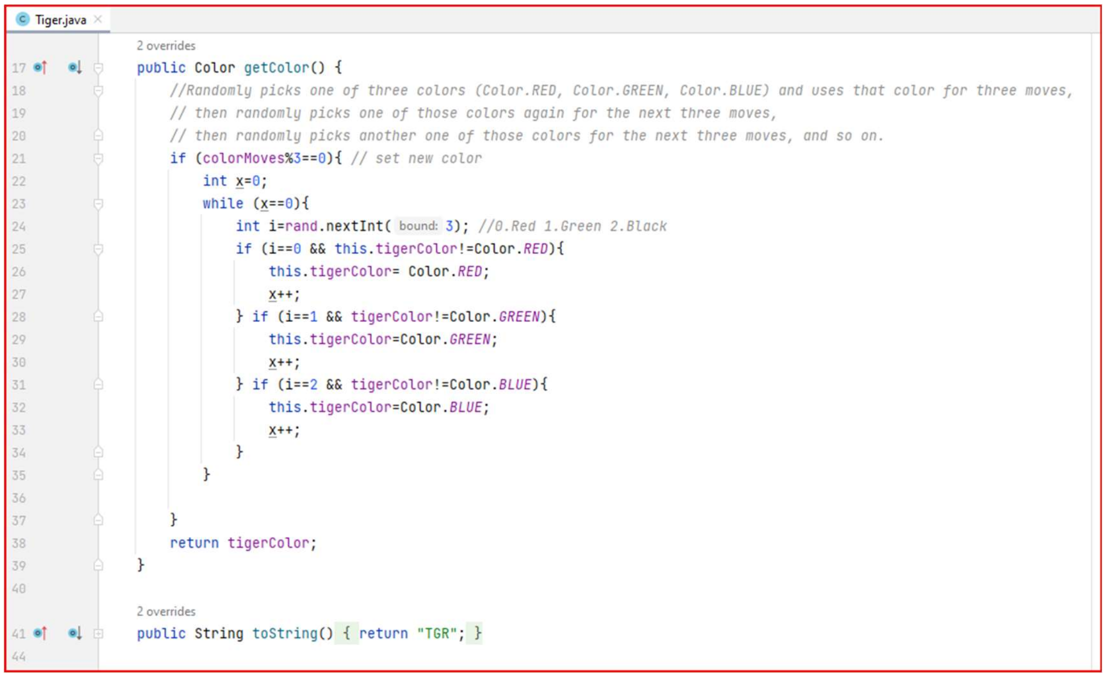
```
    public String toString(){
        //Should alternate on each different move between a slash character (/)
        // and a backslash character () starting with a slash.
        if (moves%2==0){
            return "/";
        } else {
            return "\\";
        }

    }

    4 usages
    public Action getMove(CritterInfo info){
        //always infect if an enemy is in front, otherwise hop if possible, otherwise turn left.
        moves++;
        if(info.getFront()==Neighbor.OTHER){
            return Action.INFECT;
        } else if (info.getFront()==Neighbor.EMPTY){
            return Action.HOP;
        } else {
            return super.getMove(info);
        }
    }

}
```

Tiger.java

Constructor	**public Tiger()**
getColor	Randomly picks one of three colors (Color.RED, Color.GREEN, Color.BLUE) and uses that color for three moves, then randomly picks one of those colors again for the next three moves, then randomly picks another one of those colors for the next three moves, and so on.
toString	"TGR"
getMove	Always infect if an enemy is in front, otherwise if a wall is in front or to the right, then turn left, otherwise if a fellow Tiger is in front, then turn right, otherwise hop.

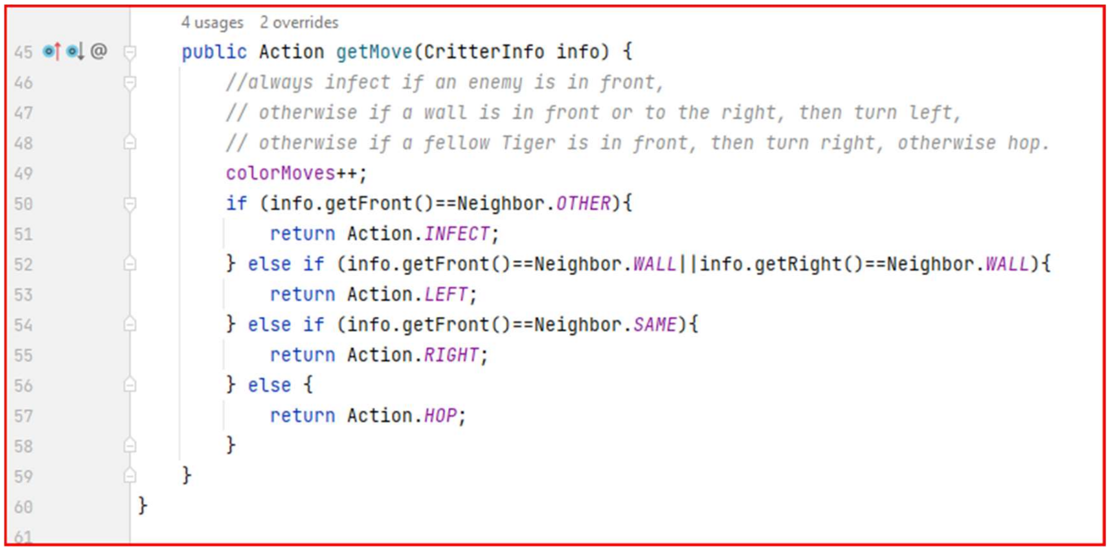


```
// * Assignment 3 | Software Design and Development (January 2023)
// * Author: Naresh Narendran UNU2200290

import java.awt.*;
import java.util.*;

3 usages   2 inheritors
public class Tiger extends Critter {
    3 usages
    private int colorMoves;
    7 usages
    Color tigerColor;
    1 usage
    Random rand = new Random();

    2 usages
    public Tiger(){
        colorMoves=0;//1,2,3
        getColor();
    }

```



Tiger.java

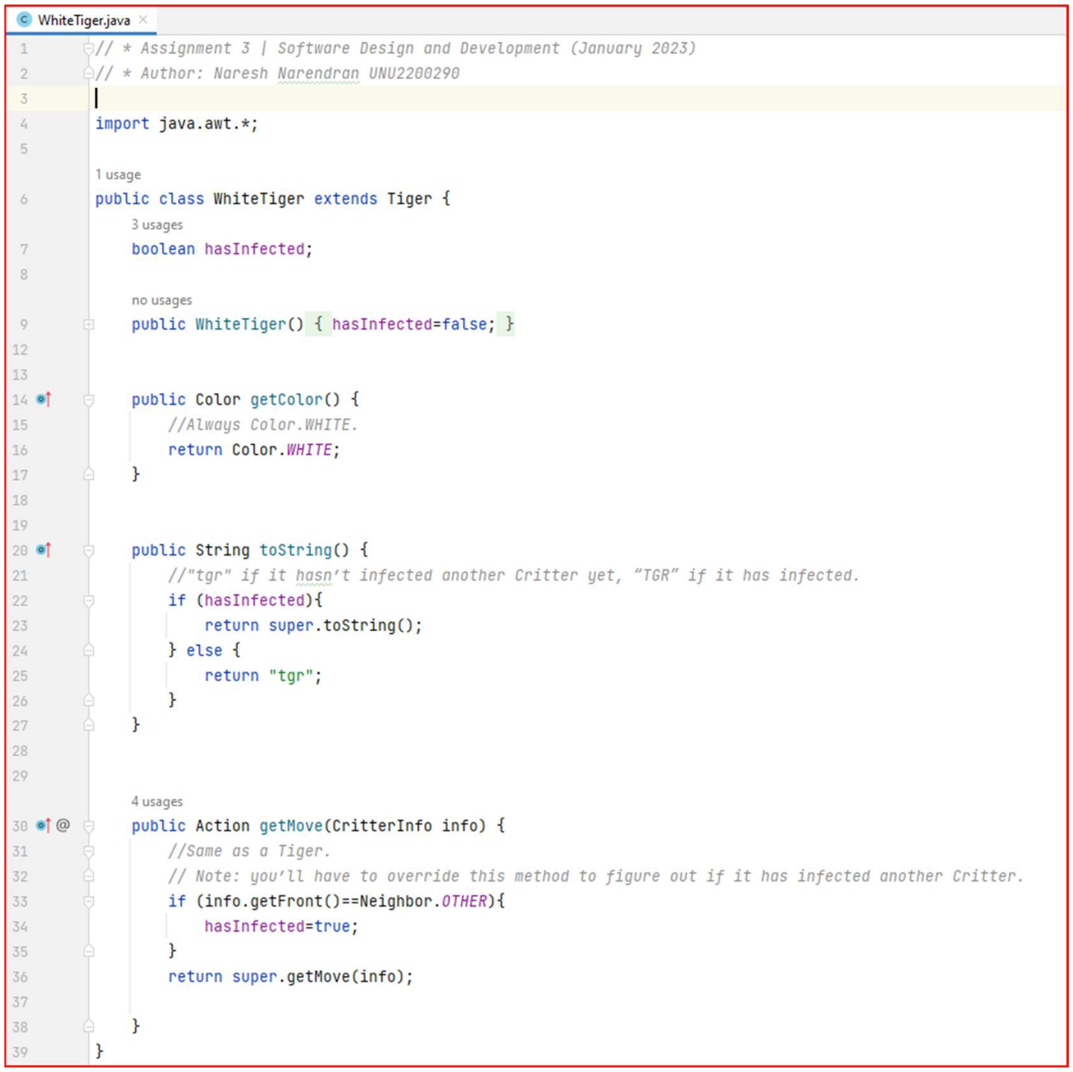
```java
    2 overrides
    public Color getColor() {
        //Randomly picks one of three colors (Color.RED, Color.GREEN, Color.BLUE) and uses that color for three moves,
        // then randomly picks one of those colors again for the next three moves,
        // then randomly picks another one of those colors for the next three moves, and so on.
        if (colorMoves%3==0){ // set new color
            int x=0;
            while (x==0){
                int i=rand.nextInt( bound: 3); //0.Red 1.Green 2.Black
                if (i==0 && this.tigerColor!=Color.RED){
                    this.tigerColor= Color.RED;
                    x++;
                } if (i==1 && tigerColor!=Color.GREEN){
                    this.tigerColor=Color.GREEN;
                    x++;
                } if (i==2 && tigerColor!=Color.BLUE){
                    this.tigerColor=Color.BLUE;
                    x++;
                }
            }

        }
        return tigerColor;
    }

    2 overrides
    public String toString() { return "TGR"; }
```



```java
    4 usages   2 overrides
    public Action getMove(CritterInfo info) {
        //always infect if an enemy is in front,
        // otherwise if a wall is in front or to the right, then turn left,
        // otherwise if a fellow Tiger is in front, then turn right, otherwise hop.
        colorMoves++;
        if (info.getFront()==Neighbor.OTHER){
            return Action.INFECT;
        } else if (info.getFront()==Neighbor.WALL||info.getRight()==Neighbor.WALL){
            return Action.LEFT;
        } else if (info.getFront()==Neighbor.SAME){
            return Action.RIGHT;
        } else {
            return Action.HOP;
        }
    }
}
```

WhiteTiger.java

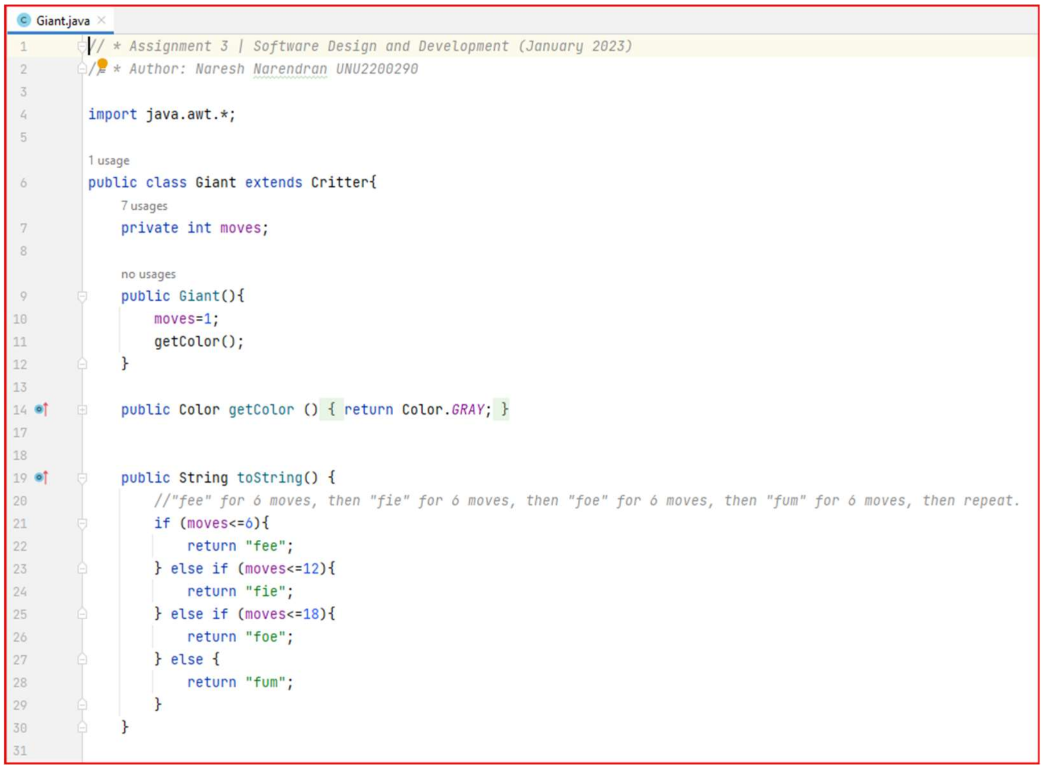
Constructor	**public WhiteTiger()**
getColor	Always Color.WHITE.
toString	"tgr" if it hasn't infected another Critter yet, "TGR" if it has infected.
getMove	Same as a Tiger. Note: you'll have to override this method to figure out if it has infected another Critter.


WhiteTiger.java

```java
// * Assignment 3 | Software Design and Development (January 2023)
// * Author: Naresh Narendran UNU2200290

import java.awt.*;

1 usage
public class WhiteTiger extends Tiger {
    3 usages
    boolean hasInfected;

    no usages
    public WhiteTiger() { hasInfected=false; }


    public Color getColor() {
        //Always Color.WHITE.
        return Color.WHITE;
    }


    public String toString() {
        //"tgr" if it hasn't infected another Critter yet, "TGR" if it has infected.
        if (hasInfected){
            return super.toString();
        } else {
            return "tgr";
        }
    }


    4 usages
    public Action getMove(CritterInfo info) {
        //Same as a Tiger.
        // Note: you'll have to override this method to figure out if it has infected another Critter.
        if (info.getFront()==Neighbor.OTHER){
            hasInfected=true;
        }
        return super.getMove(info);

    }
}
```

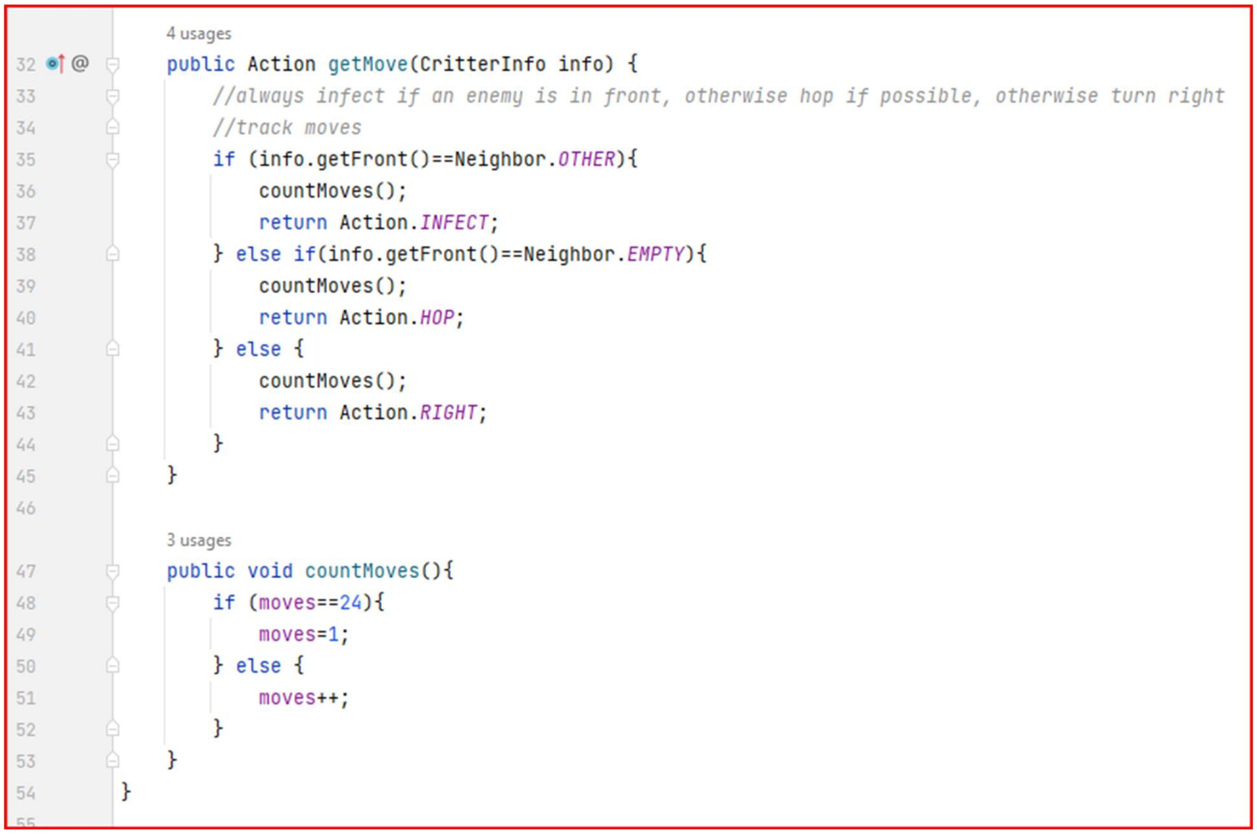
Giant.java

Constructor	**public Giant()**
getColor	Color.GRAY
toString	"fee" for 6 moves, then "fie" for 6 moves, then "foe" for 6 moves, then "fum" for 6 moves, then repeat.
getMove	always infect if an enemy is in front, otherwise hop, if possible, otherwise turn right.



Giant.java

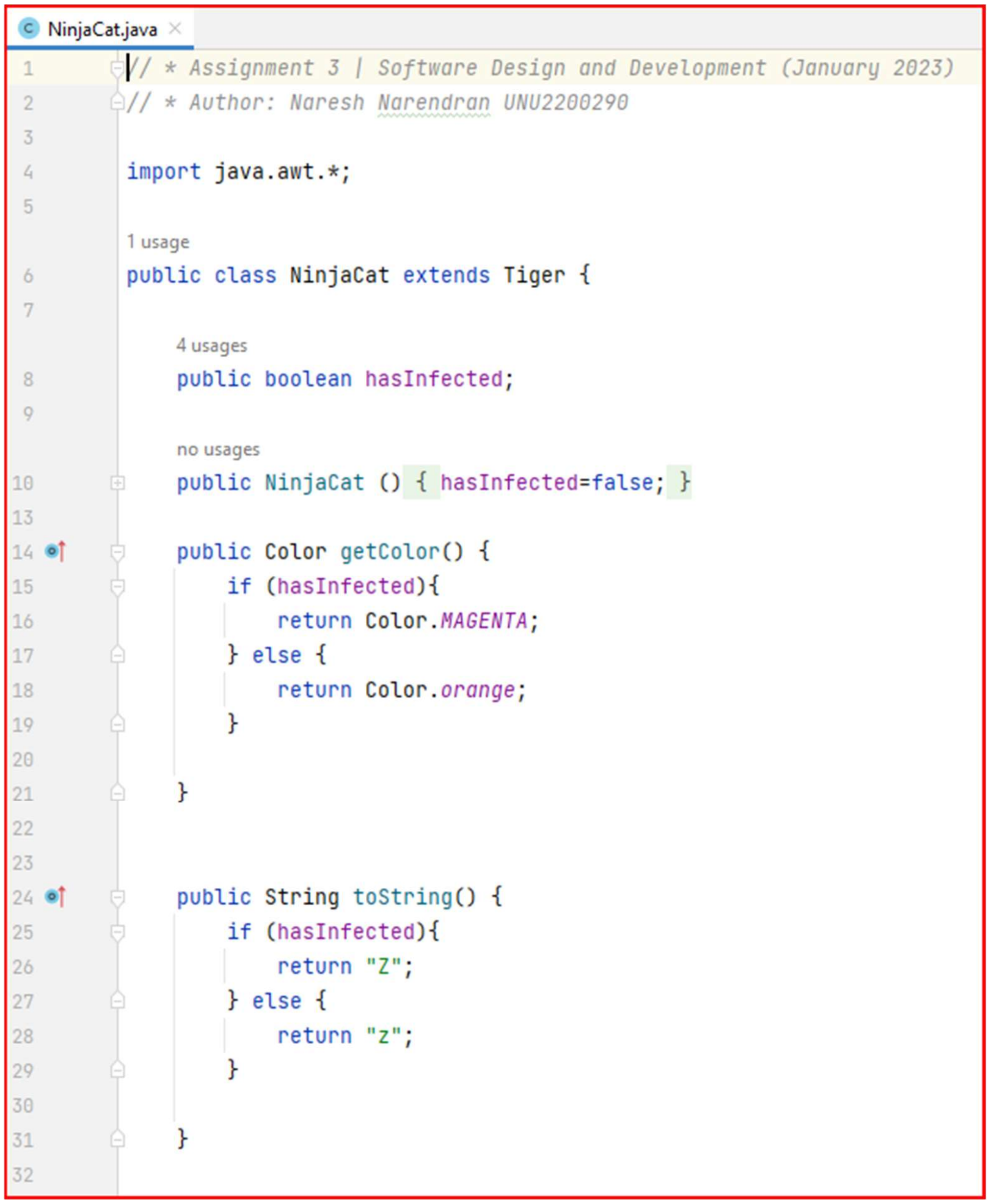
```
// * Assignment 3 | Software Design and Development (January 2023)
/* * Author: Naresh Narendran UNU2200290

import java.awt.*;

1 usage
public class Giant extends Critter{
    7 usages
    private int moves;

    no usages
    public Giant(){
        moves=1;
        getColor();
    }

    public Color getColor () { return Color.GRAY; }


    public String toString() {
        //"fee" for 6 moves, then "fie" for 6 moves, then "foe" for 6 moves, then "fum" for 6 moves, then repeat.
        if (moves<=6){
            return "fee";
        } else if (moves<=12){
            return "fie";
        } else if (moves<=18){
            return "foe";
        } else {
            return "fum";
        }
    }
```



```
    4 usages
    public Action getMove(CritterInfo info) {
        //always infect if an enemy is in front, otherwise hop if possible, otherwise turn right
        //track moves
        if (info.getFront()==Neighbor.OTHER){
            countMoves();
            return Action.INFECT;
        } else if(info.getFront()==Neighbor.EMPTY){
            countMoves();
            return Action.HOP;
        } else {
            countMoves();
            return Action.RIGHT;
        }
    }

    3 usages
    public void countMoves(){
        if (moves==24){
            moves=1;
        } else {
            moves++;
        }
    }
}
```

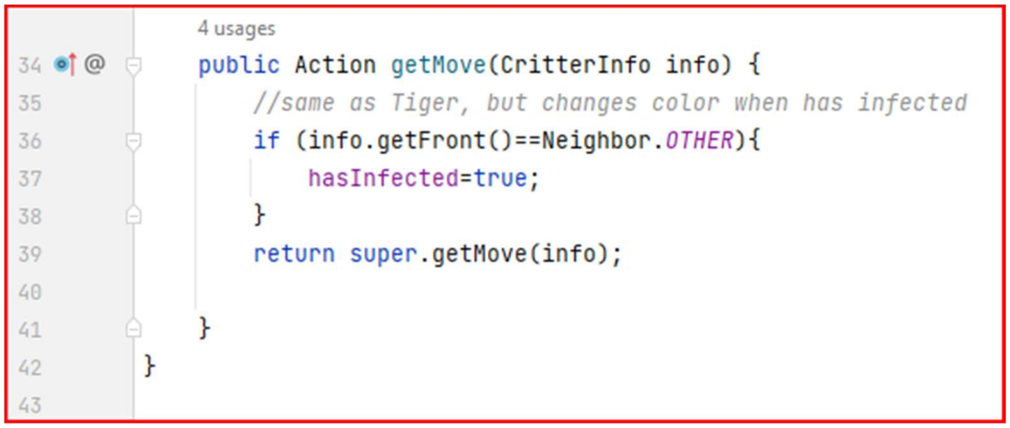
NinjaCat.java

Constructor	**public NinjaCat()**
getColor	Randomly picks one of three colors (Color.RED, Color.GREEN, Color.BLUE) and uses that color for three moves, then randomly picks one of those colors again for the next three moves, then randomly picks another one of those colors for the next three moves, and so on.
toString	“(=^.^=)”
getMove	Always infect if an enemy is in front, otherwise if a wall is in front. or to the right, then turn left, otherwise if a fellow NinjaCat is in front, then turn right, otherwise hop.



NinjaCat.java

```java
// * Assignment 3 | Software Design and Development (January 2023)
// * Author: Naresh Narendran UNU2200290

import java.awt.*;

1 usage
public class NinjaCat extends Tiger {

    4 usages
    public boolean hasInfected;

    no usages
    public NinjaCat () { hasInfected=false; }

    public Color getColor() {
        if (hasInfected){
            return Color.MAGENTA;
        } else {
            return Color.orange;
        }

    }


    public String toString() {
        if (hasInfected){
            return "Z";
        } else {
            return "z";
        }

    }

```

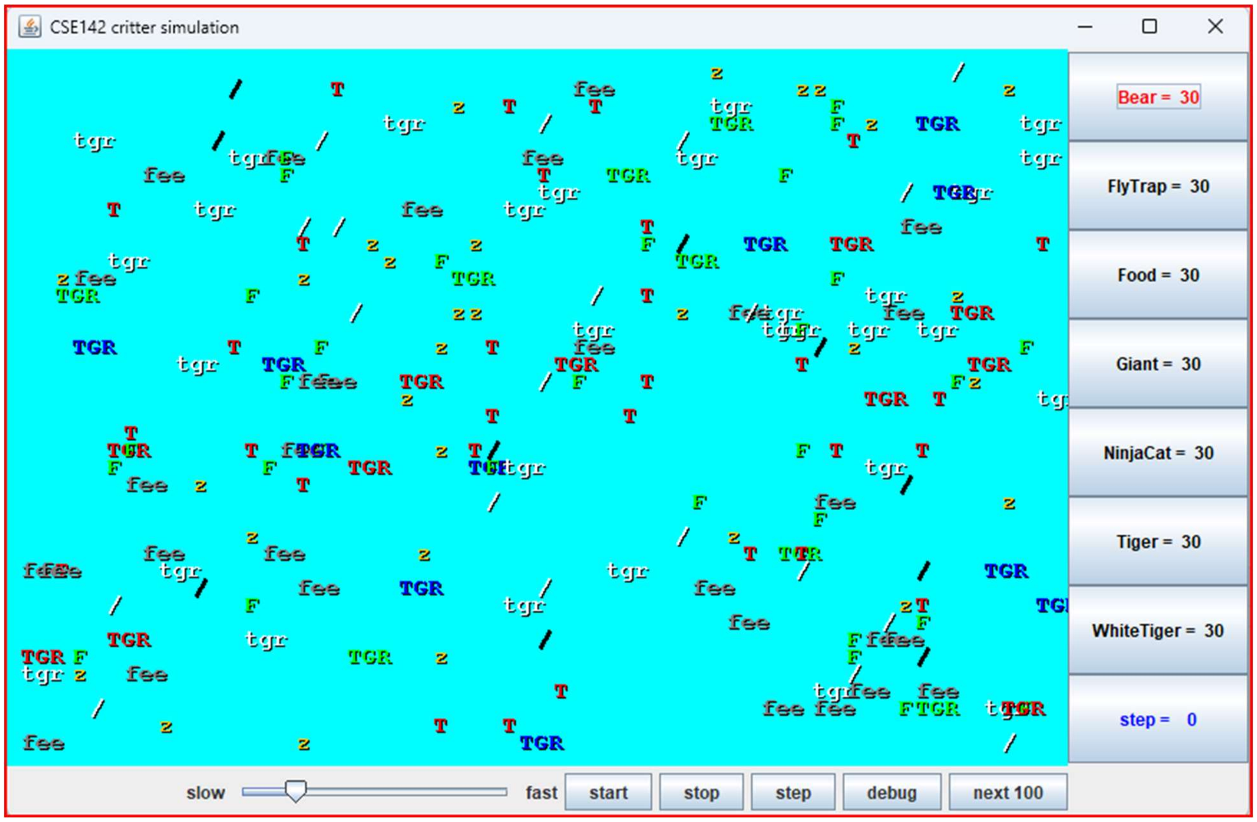


```java
    4 usages
    public Action getMove(CritterInfo info) {
        //same as Tiger, but changes color when has infected
        if (info.getFront()==Neighbor.OTHER){
            hasInfected=true;
        }
        return super.getMove(info);

    }
}

```

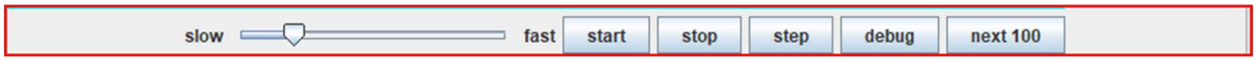
7.0 RESULTS

Running in simulation mode for the purpose of testing.

All classes have been built and successfully running. The life status is shown on the right-hand side of the dashboard. The layout is as below.

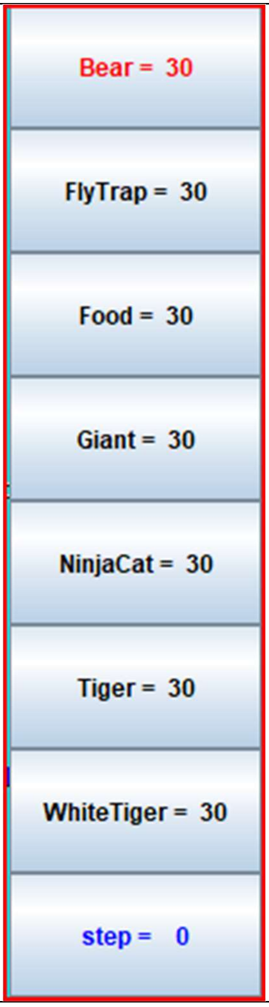


The speed function has been added to run the animal functions faster by toggle as below.



Screen	**Layout Button Explanation**
Speed	Toggle between slow/fast slider
Start	Begin the Game
Stop	Pause/Stop Completely
Debug	Stop game for checking/debugging
Next 100	Display result of next 100 steps

Count of Life Remaining	Layout Button Explanation	

30	Bear	 Bear = 30
		FlyTrap = 30
		Food = 30
		Giant = 30
		NinjaCat = 30
		Tiger = 30
		WhiteTiger = 30
		step = 0

30	FlyTrap	
30	Food	
30	Giant	
30	NinjaCat	
30	Tiger	
30	White Tiger	
	Steps Done	

8.0 DISCUSSION

This project's development was beset with formidable obstacles. Programming errors, typographical errors, and Java SDK issues. This undertaking has been successfully completed due to perseverance and a never-give-up attitude.

The project produces good programming skills, good coding practices, appropriate documentation, and minimal code by utilizing private and public methods or classes. In addition to promoting analytical thinking in the design and development of complete software, the final project provides space for the development of new versions.

In terms of the Animal Kingdom, the Java program that models it using the classes (Critter.java, Critter.Frame.java, CritterInfo, CritterMain.java, CritterModel.java, CritterPanel.java, FlyTrap.java, Food.java, Giant.java, NinjaCat.java, Tiger.java, and WhiteTiger.java) is extremely important. Here are several important details to emphasize: -

Understanding Animal Behaviors and Characteristics

The Java software offers a framework for simulating and modeling the many traits and behaviors of animals in the Animal Kingdom. The program enables the representation of animal behaviors including locomotion, interaction, and reproduction by using Java principles like classes, objects, inheritance, polymorphism, and encapsulation. This advances the study of biology and animal behavior by giving us a greater knowledge of how creatures in the Animal Kingdom might act in their natural environments.

Predictive modeling and simulations

The Java application can be used to build simulations to research animal behavior in regulated virtual settings. This can assist researchers and scientists in forecasting animal behavior in situations that occur in the real world, such as interactions between predators and prey, animal migrations, or population dynamics. These simulations can be useful tools for researching and forecasting animal behavior in circumstances when it would be impossible or expensive to do field research, providing insights into the dynamics of the Animal Kingdom.

Tools for Education

The Java application can be used to teach and learn about the animal kingdom. It can be applied in educational contexts including classrooms, colleges, and zoos to assist students in comprehending the variety and complexity of animals, as well as their traits and behaviors. The program's interactive features can involve students in a hands-on learning experience, allowing them to investigate and comprehend animal behaviors in a real-world setting.

Models for Science

The Java software can also be used as a model for science to research animal behavior in a lab setting. The program can be used by researchers to test theories, plan studies, and examine data pertaining to animal behavior. The program is a useful tool for scientific studies in the areas of biology, ecology, and animal behavior since it can be altered and expanded to represent animal species or behaviors of interest.

Future Developments Possibility

The Java software can be improved and enhanced to accommodate more sophisticated animal behaviors, different animal species, or new research questions. It can serve as a starting point for the development of further simulations, teaching resources, or research models that are more complex. Because of Java's versatility and extensibility, the program could be improved upon and used in a variety of animal kingdom-related domains.

In conclusion, the Java application that simulates the Animal Kingdom has a tremendous impact on helping us grasp the variety of traits and behaviors that animals exhibit. It can be applied in a variety of ways to research animal behaviors, forecast outcomes, and offer insights into the workings of the Animal Kingdom, such as simulations, teaching aids, or scientific models. The application has the potential for future improvements and changes, making it a useful tool for studying and teaching animal behavior.

9.0 CONCLUSION

This undertaking will cultivate Object Oriented Programming (OOP) using the Java programming language. The project aims to improve programming skills by utilizing OOP techniques in Java Classes and Methods to create the Animal Kingdom Game.

In conclusion, the Animal Kingdom is a vast and complex group of living things that are extremely important in many academic subjects and play a crucial part in ecosystems. Using object-oriented programming ideas like classes, objects, inheritance, polymorphism, and encapsulation to model the animal kingdom in Java offers a strong framework for describing the various traits and behaviors of animals.

During this presentation, we covered the presentation's goals and purpose, introduced a Java program that was relevant to the animal kingdom, explained the program's use, functionality, and importance in relation to the animal kingdom, and covered the major Java concepts that were applied to the program. Additionally, we included examples and demos of the program's functionality to explain how objects are used to represent animals, how Java classes and methods are used to model their properties, and how the software replicates animal behaviors.

We stressed the diversity and complexity of the Animal Kingdom as well as the advantages of modeling it in Java, such as the capacity to capture the numerous facets of animal biology, ecology, and behavior in a systematic and organized manner. The object-oriented philosophy of Java, which includes elements like classes, objects, inheritance, polymorphism, and encapsulation, offers a powerful and adaptable foundation for describing the various traits and behaviors of animals in a modular and upkeep-friendly manner.

Applications of Java modeling to the study of the animal kingdom include biology, ecology, conservation, education, and entertainment. We can develop sophisticated and accurate simulations, teaching aids, or scientific models that assist us in comprehending and appreciating the complexity and diversity of the Animal Kingdom by utilizing the power of Java programming techniques.

We hope you now have a better understanding of the interesting world of animals and how Java may be used to accurately portray their traits and activities.