|  |  |
| --- | --- |
| Clyde Conservation  Technical Documents | Containing  Project brief, designs, code, test logs and screen dumps.  Robert Lothian  HND Software Development 2 |

CONTENTS

Project Brief – Page 2

Designs – Page 3

Code – Page 6

Test Logs – Page 18

Screen Dumps – Page 26

Project Brief

The assignment was to create a program which will log animal details, assign keepers to cages and keep track of the space left available to allocate within these cages. I will now break each area down in further detail.

*Logging Animals*

Clyde Conservation currently keeps mammal’s and reptiles, and as such we needed two options for the user to choose from when assigning details to an animal. For mammals’, information is prompted for input from the user regarding the animal’s name, the type of animal, its mates name, sex, if it is female if it has given birth or not and finally a danger rating, this rating would be used to go on to calculate the cost of insurance. For reptiles we must prompt the user for the animal’s name, type of animal, sex, danger rating, tank temperature and environment (wet or dry).

As there is information requested by both mammals and reptiles it makes sense to create a parent class of Animal, which we can assign an incremental ID for cataloguing purposes, to store these details and then use inheritance to pass this information onto child classes of Reptile and Mammal.

In our Reptile and Mammal classes we have created a formula to calculate insurance costs, this had to be done within the child classes as the formula differs depending on whether the animal is a Reptile or Mammal.

To store this information we need to create an Array List that will keep this information stored so it can be accessed later, going forward this information can be converted into a data table and stored permanently.

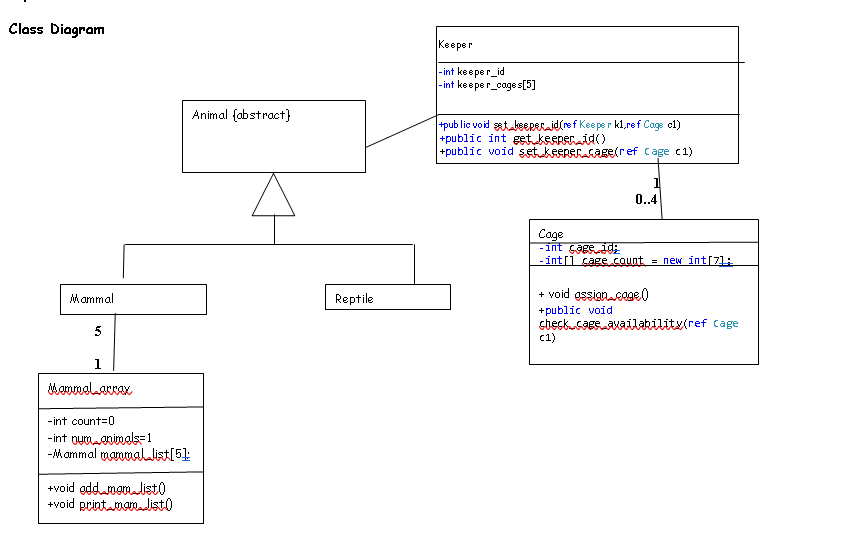
*Cage assignments*

The other major part of our brief is the assignment of cages to keepers. Clyde Conservation has a finite number of keepers, cages and spaces within these cages (each cage having a different capacity) and as such we needed to formulate a solution that can easily allocate these resources without exceeding their limits.

To do this we need to create two classes that will interact, one for the Keeper and one for Cages. We are going to use arrays to define the number of keepers, cages and spaces available. We need to ensure that if the limit of Keepers or Cages is exceeded a message is displayed, we also need to employ the use of conditional statements so that the spaces available within the cages cannot be less than 0.

Designs

The following are the designs that were provided with the project.



|  |  |
| --- | --- |
| **Method Name: Mammal\_list (constructor)** | |
| **Inputs** | |
| **Parameters:**  None | **User input:**  None |
| **Outputs** | |
| **Return type:**  None | **Output**  None |
| **Process(es)**  **Description**:  No argument, constructor to initialise all fields  **Pseudo-code ( attach a separate sheet if necessary ):**  **Level 1**  1. set count to 0  2. set number\_animals to 1 | |

|  |  |
| --- | --- |
| **Method Name: Add\_mammal\_list** | |
| **Inputs** | |
| **Parameters:**  None | **User input:**  None |
| **Outputs** | |
| **Return type:**  Void | **Output**  Warning message |
| **Process(es)**  **Description**:  Searches for a space in the array of objects (id = 0). If it is found, adds the animal and mammal details to the available element of the array.  **Pseudo-code ( attach a separate sheet if necessary ):**  Level 1  1. clear the screen  2. loopwhile search through mammal array for an id(call get\_id function) of 0 and limit loop to 5  Loopend  3. decrement count variable by 1  4. Add mammal and animal details to array list  **Level 2**  1. clear the screen  2. loopwhile search through mammal array for an id(call get\_id function) of 0 and limit loop to 5  Loopend  3. decrement count variable by 1  4.1 if count < 5 and mammal array id equals 0  4.2 call get\_mammal\_details to read into the array  4.3 add 1 to the number\_animals counter  Else  4.4 print capacity of array exceeded message  Endif | |

|  |  |
| --- | --- |
| **Method Name: show\_mammal\_list** | |
| **Inputs** | |
| **Parameters:**  None | **User input:**  None |
| **Outputs** | |
| **Return type:**  Void | **Output**  Each element in the array of objects for mammal |
| **Process(es)**  **Description**:  Checks each element in the array of objects, if the element has an id that is not 0, it will print the contents.  **Pseudo-code ( attach a separate sheet if necessary ):**  **Level 1**  1. Clear the screen  2. loopfor 0 to 5  3. if counter < 5 and and element in array of objects does not equal 0  4. print the element by calling print\_mammal\_details  Endif  Loopend | |

Code

A copy of the code will be provided in a zip file along with documentation

*Program*

//Robert Lothian

//HND Software Development 2

//03.12.21

//Assessment 1, Clyde Conservation program

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Assessment\_1

{

class Program

{

static void Main(string[] args)

{

//instantiate a reptile and mammal object - named m1 and r1. Mammal array - m1. Keeper - k1. Cage - c1

Mammal m1 = new Mammal();

Reptile r1 = new Reptile();

Mammal\_array ma = new Mammal\_array();

Cage c1 = new Cage();

Keeper k1 = new Keeper();

Keeper kp = new Keeper();

const int quit\_option = 4;

int choice = 0;

int num\_animals = 0;

while (choice != quit\_option)

{

Console.WriteLine();

display\_menu();

get\_choice(ref choice);

act\_on\_choice(ref num\_animals, ref m1, ref r1, choice, ma, k1, c1, kp);

}

}

public static void display\_menu()

//displays the menu system

{

Console.WriteLine();

Console.WriteLine("\t\t\t MAIN MENU");

Console.WriteLine("\t\t\t 1. Mammal");

Console.WriteLine ("\t\t 2. Allocate Keeper to Cage");

Console.WriteLine("\t\t\t 3. Reptile");

Console.WriteLine("\t\t\t 4. Exit ");

} //end of main menu

public static void get\_choice(ref int this\_choice)

{

Console.WriteLine("\t\t Enter Choice 1,2, 3 or 4\n");

this\_choice = int.Parse(Console.ReadLine());

} //end of get choice method

public static void act\_on\_choice(ref int num\_animals, ref Mammal m1, ref Reptile r1, int this\_choice, Mammal\_array ma, Keeper k1, Cage c1, Keeper kp)

{

//Takes user input from main menu and calls Mammal or Reptile . This in turn will call the methods for

//Mammal or reptile

switch (this\_choice)

{

case 1:

{

int Mchoice;

Console.WriteLine();

Console.WriteLine("\t\t\tSelect operation \n");

Console.WriteLine("\t\t\t1. Add mammal \n");

Console.WriteLine("\t\t\t2. Print mammal \n");

Console.WriteLine("\t\t\tEnter choice: ");

Mchoice = int.Parse(Console.ReadLine());

switch (Mchoice)

{

case 1:

{

//call mammal list

ma.Add\_mammal\_list();

Console.WriteLine("Press any key to continue....");

Console.ReadKey();

break;

}

case 2:

{

//print mammal list

ma.show\_mammal\_list();

Console.WriteLine("Press any key to continue....");

Console.ReadKey();

break;

}

}//end switch statement for mammal type

break;

} // end case 1 for main menu

case 2:

k1.set\_keeper\_id(ref k1, ref c1);

break;

case 3:

{

int Rchoice;

Console.WriteLine();

Console.WriteLine("\t\t\tSelect operation \n");

Console.WriteLine("\t\t\t1. Add Reptile \n");

Console.WriteLine("\t\t\t2. Print Reptile \n");

Console.WriteLine("\t\t\tEnter choice: ");

Rchoice = int.Parse(Console.ReadLine());

Console.WriteLine("\n");

switch (Rchoice)

{

case 1:

{

//call set reptile details from reptile class

r1.set\_reptile\_details(ref num\_animals);

Console.WriteLine("Press any key to continue....");

Console.ReadKey();

break;

}

case 2:

{

//call print reptile details from reptile class

r1.print\_reptile\_details();

Console.WriteLine("Press any key to continue....");

Console.ReadKey();

break;

}

}//end switch statement for Reptile type

break;

} //end of case 3

case 4:

break;

}//end of switch

} //end of act on choice method

}//end of class

}

*Animal Class*

//Robert Lothian

//HNS Software Development 2

//03.12.21

//Assesment 1

using System;

using System.Collections.Generic;

using System.Text;

namespace Assessment\_1

{

class Animal //this creates a parent object that contains information to be inherited by Mammal and Reptile classes

{

private protected int id;

private protected int danger\_rating;

private protected string animal\_name;

private protected string animal\_type;

private protected char sex;

public Animal() //this is a constructor that creates default values for each object of Animal created

{

id = 0;

animal\_name = "";

animal\_type = "";

danger\_rating = 0;

sex = 'F';

}

public void set\_animal\_details(ref int num\_animals) //this method sets the deatils for the animal by asking for user input and storing detail

{

id++; //increment id

Console.Write("Enter animals name - "); //animal name

animal\_name = Console.ReadLine();

Console.Write("Enter animals type - "); //animal type

animal\_type = Console.ReadLine();

Console.Write("Enter animals sex - "); //animal sex

sex = char.Parse(Console.ReadLine().ToUpper());

while (sex != 'F' && sex != 'M')

{

Console.WriteLine("You have entered a wrong character, try again"); //this will loop if user enters invalid char

Console.Write("Enter animals sex - ");

sex = char.Parse(Console.ReadLine().ToUpper());

}

Console.Write("Enter animals danger rating - "); //danger rating

danger\_rating = int.Parse(Console.ReadLine());

while (danger\_rating > 5 || danger\_rating <= 0)

{

Console.WriteLine("You have entered an invalid rating, try again"); //this will loop if user enters an invalid damage rating

Console.Write("Enter animals danger rating - ");

danger\_rating = int.Parse(Console.ReadLine());

}

}

public int get\_id() //the following sections get user entered results

{

return id;

}

public string get\_animal\_name()

{

return animal\_name;

}

public string get\_animal\_type()

{

return animal\_type;

}

public int get\_danger\_rating()

{

return danger\_rating;

}

public char get\_sex()

{

return sex;

} //get-ers end

public void print\_animal\_details() //this prints out the data the user entered

{

Console.WriteLine("ANIMAL ID - " + get\_id());

Console.WriteLine("ANIMAL NAME - " + get\_animal\_name());

Console.WriteLine("ANIMAL TYPE - " + get\_animal\_type());

Console.WriteLine("SEX - " + get\_sex());

Console.WriteLine("DANGER RATING - " + get\_danger\_rating());

}

public virtual double calculate\_insurance(int danger\_rating) //this sets up the calculations of insurance. This will be changed in the child classes by polymorphism

{

int base\_insurance = 0;

double insurance = 0;

switch (danger\_rating)

{

case 1:

insurance = (base\_insurance \* 10) \* 0.02;

break;

case 2:

insurance = (base\_insurance \* 20) \* 0.05;

break;

case 3:

insurance = (base\_insurance \* 30) \* 0.05;

break;

case 4:

insurance = (base\_insurance \* 40) \* 0.05;

break;

case 5:

insurance = (base\_insurance \* 50) \* 0.10;

break;

}

return insurance;

}

} //end of class Animal

}//end of namespace

*Mammal Class*

//Robert Lothian

//HNS Software Development 2

//03.12.21

//Assesment 1

using System;

using System.Collections.Generic;

using System.Text;

namespace Assessment\_1

{

class Mammal : Animal //this creates a child class that will inherit from the parent Animal class

{

private protected string mate\_name;

private protected string given\_birth;

public Mammal() //this instanciates the object Mammal with default values

{

mate\_name = "";

given\_birth = "";

}

public void set\_mammal\_details(ref int num\_animals)

{

set\_animal\_details(ref num\_animals);

Console.Write("Enter the mates name : ");

mate\_name = Console.ReadLine();

if (sex == 'F')

{

Console.Write("Has the animal given birth? : ");

given\_birth = Console.ReadLine();

}//end of if statement

else given\_birth = "";

}

public string get\_mate\_name()

{

return mate\_name;

}

public string get\_given\_birth()

{

return given\_birth;

}

public void print\_mammal\_details()

{

print\_animal\_details();

Console.WriteLine("MATE NAME: " + mate\_name);

if (sex == 'F') //this if statement will ask if mammal has given birth if sex = f

{

Console.WriteLine("GIVEN BIRTH: " + given\_birth);

}

Console.WriteLine("INSURANCE COST: £" + calculate\_insurance(danger\_rating));

}

public override double calculate\_insurance(int danger\_rating) //calculates insurance based on danger rating

{

int base\_insurance = 5000;

double insurance = 0;

switch (danger\_rating)

{

case 1:

insurance = (base\_insurance \* 10) \* 0.02;

break;

case 2:

insurance = (base\_insurance \* 20) \* 0.05;

break;

case 3:

insurance = (base\_insurance \* 30) \* 0.05;

break;

case 4:

insurance = (base\_insurance \* 40) \* 0.05;

break;

case 5:

insurance = (base\_insurance \* 50) \* 0.10;

break;

}

return insurance;

}

}

}

*Reptile Class*

//Robert Lothian

//HNS Software Development 2

//03.12.21

//Assesment 1

using System;

namespace Assessment\_1

{

class Reptile : Animal

{

private protected float tank\_temperature;

private protected string enviroment;

public Reptile() //instanciates reptile attributes

{

tank\_temperature = 0;

enviroment = "";

}

public void set\_reptile\_details(ref int num\_animals) //sets reptile sprecific details

{

set\_animal\_details(ref num\_animals);

Console.Write("Enter the tank temperature - ");

tank\_temperature = float.Parse(Console.ReadLine());

Console.Write("Enter enviroment - water or dry - ");

enviroment = Console.ReadLine();

}

public double get\_tank\_temperature()

{

return tank\_temperature;

}

public string get\_enviroment()

{

return enviroment;

}

public void print\_reptile\_details()

{

print\_animal\_details();

Console.WriteLine("TANK TEMPERATURE: " + get\_tank\_temperature() + "C");

Console.WriteLine("ENVIROMENT: " + get\_enviroment());

Console.WriteLine("INSURANCE COST: £" + calculate\_insurance(danger\_rating));

}

public override double calculate\_insurance(int danger\_rating) //override base animal danger rating using polymorphism

{

int base\_insurance = 1000;

double insurance = 0;

switch (danger\_rating)

{

case 1:

insurance = (base\_insurance \* 10) \* 0.05;

break;

case 2:

insurance = (base\_insurance \* 20) \* 0.10;

break;

case 3:

insurance = (base\_insurance \* 30) \* 0.15;

break;

case 4:

insurance = (base\_insurance \* 40) \* 0.15;

break;

case 5:

insurance = (base\_insurance \* 50) \* 0.20;

break;

}

return insurance;

}

}// end of class

}//end of namespace

*Mammal Array Class*

//Robert Lothian

//HND Software Development 2

//11.02.22

//This class is used to create an array list

using System;

using System.Collections.Generic;

using System.Text;

namespace Assessment\_1

{

class Mammal\_array

{

private int count = 0;

int num\_animals = 0;

Mammal[] mammal\_list = new Mammal[5];

public Mammal\_array()

{

count = 0; //counter for array

num\_animals = 5; //max amount in array

for (int i=0; i< 5; i ++) //loop to instanciate array

{

mammal\_list[i] = new Mammal(); //create mammal\_list

}

}

public void Add\_mammal\_list()

{

if (count <= num\_animals)

{

mammal\_list[count].set\_mammal\_details(ref num\_animals);

mammal\_list[count].get\_id();

count++;

}

else

{

Console.WriteLine("Cannot add, array is full");

}

}

public void show\_mammal\_list() //this loop prints out mammal details

{

for (int i = 0; i <num\_animals; i++)

{

if ((mammal\_list[i].get\_id() !=0))

{

mammal\_list[i].print\_mammal\_details();

Console.WriteLine("\n");

}

}//end loop

}

}

}

*Keeper Class*

//Robert Lothian

//HND Software Development 2

//11.02.22

//This class is used to create keepers

using System;

using System.Collections.Generic;

using System.Text;

namespace Assessment\_1

{

class Keeper

{

int keeper\_id;

int count;//added count to cages taken by keeper

int[] cages\_allocated = new int[4]; //sets up to 4 keepers

public Keeper()

{

keeper\_id = 0;

cages\_allocated[0] = 0;

cages\_allocated[1] = 0;

cages\_allocated[2] = 0;

cages\_allocated[3] = 0;

}

public void set\_keeper\_id(ref Keeper k1, ref Cage c1)

{

//user will input keeper to be assigned to a cage

Console.WriteLine("Enter the Keeper ID that requires a cage: ");

keeper\_id = int.Parse(Console.ReadLine());

keeper\_id--;

set\_keeper\_cage(ref c1);

}

public int get\_keeper\_id()

{

keeper\_id++;

return keeper\_id;

}

public void set\_keeper\_cage(ref Cage c1)

{

if (cages\_allocated[keeper\_id] + 1 >= 0 && cages\_allocated[keeper\_id] + 1 <= 4) //this statement determines if there is available space

{

c1.check\_cage\_availability(ref c1);

cages\_allocated[keeper\_id]++;

count= cages\_allocated[keeper\_id];//keeps count of cages taken by keeper

Console.WriteLine("Keeper " + get\_keeper\_id() + " has " + count + " cage(s)");

}

else

{

Console.WriteLine("There are no available cages");

}

}

}//end class

}

//Robert Lothian

//HND Software Development 2

//11.02.22

//This class is used to create cages for the animals

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Assessment\_1

{

class Cage

{

private int cage\_id;

int[] cage\_count = new int[7]; //array to set the number of cages

public Cage()//initialise variables

{

cage\_count[0] = 4;

cage\_count[1] = 2;

cage\_count[2] = 2;

cage\_count[3] = 1;

cage\_count[4] = 3;

cage\_count[5] = 2;

cage\_count[6] = 1;

}

public void assign\_cage()

{

Console.WriteLine("Enter cage ID to be allocated");

cage\_id = int.Parse(Console.ReadLine());

cage\_id--; //take one away from cage ID as arrays start at 0

cage\_count[cage\_id] = cage\_count[cage\_id] - 1; //updates cage count

if (cage\_count[cage\_id] < 0) //this if statement determines if there are spaces left in the cage

{

cage\_count[cage\_id] = 0;

Console.WriteLine("There are no spaces left");

}

else

{

Console.WriteLine("Remaining spaces in cage " + (cage\_id + 1) + " = " + (cage\_count[cage\_id])); //changed display text to reflect spaces within a cage available rather than cages as a whole,

}

}

public void check\_cage\_availability(ref Cage c1)//called from keeper class

{

for (int i=0;i<7;i++)

{

Console.WriteLine("Cage "+(i+1)+" has "+c1.cage\_count[i] + " spaces\n");

}

c1.assign\_cage();

}

}//end class

}

*Cage Class*

//Robert Lothian

//HND Software Development 2

//11.02.22

//This class is used to create cages for the animals

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Assessment\_1

{

class Cage

{

private int cage\_id;

int[] cage\_count = new int[7]; //array to set the number of cages

public Cage()//initialise variables

{

cage\_count[0] = 4;

cage\_count[1] = 2;

cage\_count[2] = 2;

cage\_count[3] = 1;

cage\_count[4] = 3;

cage\_count[5] = 2;

cage\_count[6] = 1;

}

public void assign\_cage()

{

Console.WriteLine("Enter cage ID to be allocated");

cage\_id = int.Parse(Console.ReadLine());

cage\_id--; //take one away from cage ID as arrays start at 0

cage\_count[cage\_id] = cage\_count[cage\_id] - 1; //updates cage count

if (cage\_count[cage\_id] < 0) //this if statement determines if there are spaces left in the cage

{

cage\_count[cage\_id] = 0;

Console.WriteLine("There are no spaces left");

}

else

{

Console.WriteLine("Remaining spaces in cage " + (cage\_id + 1) + " = " + (cage\_count[cage\_id])); //changed display text to reflect spaces within a cage available rather than cages as a whole,

}

}

public void check\_cage\_availability(ref Cage c1)//called from keeper class

{

for (int i=0;i<7;i++)

{

Console.WriteLine("Cage "+(i+1)+" has "+c1.cage\_count[i] + " spaces\n");

}

c1.assign\_cage();

}

}//end class

}

Test Logs

# Test Log Sheet

**Student Robert Lothian Project Title Clyde Conservation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Subroutine Name | **Test Case** | **Test Data** | **Expected Result** | **Actual Result** | **Comments** |
|  |  |  |  |  |  |
| Add\_mammal | Valid entry and write to file  ( optional ) | Name = George  Type = Rabbit  Danger rating = 1  Sex = Male  Name of mate = Georgina  Given birth - N | Added to mammal list array element 0. | Added to array | Program doesn’t ask if given birth as animal is male |
|  |  |  |  |  |  |
| Print\_mammal | Valid print | Contents of array from above test case | Print following:  ID = 1  Name = George  Type = Rabbit  Danger rating = 1  Sex = Male  Name of mate = Georgina  Insurance = 10,000 | ANIMAL ID - 1  ANIMAL NAME - George  ANIMAL TYPE - Rabbit  SEX - M  DANGER RATING - 1  MATE NAME: Georgina  INSURANCE COST: £1000 | The expected result insurance quote is wrong |
|  |  |  |  |  |  |
| Add\_mammal | Valid entry | Name = Penny  Type = Horse  Danger rating = 2  Sex = Female  Name of mate = Peter  Given birth = Y | Added to mammal list element 1 | Added to array | As expected |
|  |  |  |  |  |  |
| Print\_mammal | Valid print | Contents of array from above test cases (2 mammals) | Print George mammal details (above) and Penny mammal details:  ID = 2  Name = Penny  Type=Horse  Danger rating = 2  Sex = Female  Name of mate = Peter  Insurance = 20000 | Print George details then  ANIMAL ID - 1  ANIMAL NAME - Penny  ANIMAL TYPE - Horse  SEX - F  DANGER RATING - 2  MATE NAME: Peter  GIVEN BIRTH: Y  INSURANCE COST: £5000 | ID does not increment and |
| Add\_mammal | Valid entry | Name = Crazy Jack  Type = Marmoset  Danger rating = 3  Sex = Male  Name of mate = Hannah | Added to mammal list element array 2 | Added to array | As expected |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Print\_mammal | Valid print | Contents of array from above test cases (3 mammals) | Print George mammal details (above) and Penny mammal details above and Crazy Jack:  ID = 3  Name = Crazy Jack  Type = Marmoset  Danger rating = 3  Sex = Male  Name of mate = Hannah  Insurance = 1,500 | Print George and Pennyn then  ANIMAL ID - 1  ANIMAL NAME - Crazy Jack  ANIMAL TYPE - Marmoset  SEX - M  DANGER RATING - 3  MATE NAME: Hannah  INSURANCE COST: £7500 | ID does not increment and insurance cost different from expected |
|  |  |  |  |  |  |
| Add\_mammal | Valid entry | Name = Andy  Type = Ape  Danger rating =4  Sex = Male  Name of mate = Mattie  Given birth = N | Added to mammal list element array 3 | Added to array | Given birth option not available as animal is male |
|  |  |  |  |  |  |
| Print\_mammal | Valid print | Contents of array from above test cases (4 mammals) | Print George mammal details (above) and Penny mammal details,Crazy Jack details and Andy:  ID = 4  Name = Andy  Type = Ape  Danger rating =4  Sex = Male  Name of mate = Mattie  Insurance = 2000 | Print George, Penny, Crazy Jack then  ANIMAL ID - 1  ANIMAL NAME - Andy  ANIMAL TYPE - Ape  SEX - M  DANGER RATING - 4  MATE NAME: Mattie  INSURANCE COST: £10000 | ID doesn’t increase.  Insurance different from expected result |
|  |  |  |  |  |  |
| Add\_mammal | Valid entry | Name = Shilpa  Type = Tiger  Danger rating =5  Sex = Female  Name of mate = Deepak  Given birth = Y | Added to mammal list element array 4 | Added to array | As expected |
|  |  |  |  |  |  |
| Print\_mammal | Valid print | Contents of array from above test cases (5 mammals) | Print George mammal details (above) and Penny mammal details,Crazy Jack details, Andy details and Shilpa details:  ID = 5  Name = Shilpa  Type = Tiger  Danger rating =5  Sex = Female  Name of mate = Deepak  Insurance = 5000 | Print George, Penny, Crazy Jack, Andy and  ANIMAL ID - 1  ANIMAL NAME - Shilpa  ANIMAL TYPE - Tiger  SEX - F  DANGER RATING - 5  MATE NAME: Deepak  GIVEN BIRTH: Y  INSURANCE COST: £25000 | ID doesn’t increase and insurance cost different |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Add\_mammal | Invalid entry  A 6th mammal to be added. Array over capacity | Details of 1 more mammal | No space left in list message | System.IndexOutOfRangeException: 'Index was outside the bounds of the array.' | Message didn’t display, program crashed with out of bounds exception |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Allocate Keeper to cage | Valid cages | Keeper id = 1  Cage = 1 | 3 cage 1’s left | Remaining spaces in cage 1 = 3  Keeper 1 has 1 cage(s) | As expected |
|  |  |  |  |  |  |
| Allocate Keeper to cage | Valid cages | Keeper id = 1  Cage =2 | 1 cage 2’s left | Remaining spaces in cage 2 = 1  Keeper 1 has 2 cage(s) | As expected |
|  |  |  |  |  |  |
| Allocate Keeper to cage | Valid cages | Keeper id = 1  Cage =3 | 1 cage 3 left | Remaining spaces in cage 3 = 1  Keeper 1 has 3 cage(s) | As expected |
|  |  |  |  |  |  |
| Allocate Keeper to Cage | Valid cages | Keeper id = 1  Cage =4 | 0 cage 4 left | Remaining spaces in cage 4 = 0  Keeper 1 has 4 cage(s) | As expected |
|  |  |  |  |  |  |
| Allocate Keeper to Cage | Maximum cages for keeper | Keeper id = 1  Cage = 1 | Keeper cannot be allocated any more cages. | Enter the Keeper ID that requires a cage:  1  There are no available cages | As expected |
|  |  |  |  |  |  |
| Allocate Keeper to Cage | Valid cages | Keeper id = 2  Cage = 1 | 2 cage 1’s left | Remaining spaces in cage 1 = 2  Keeper 2 has 1 cage(s) | As expected |
|  |  |  |  |  |  |
| Allocate Keeper to Cage | Valid cages | Keeper id = 3  Cage = 1 | 1 cage 1 left | Remaining spaces in cage 1 = 1  Keeper 3 has 1 cage(s) | As expected |
|  |  |  |  |  |  |
| Allocate Keeper to Cage | Valid cages | Keeper id = 4  Cage = 1 | 0 cage 1 left | Remaining spaces in cage 1 = 0  Keeper 4 has 1 cage(s) | As expected |
|  |  |  |  |  |  |
| Allocate Keeper to Cage | Valid cages | Keeper id = 5  Cage = 1 | No more cage 1’s left – allocate another cage. | System.IndexOutOfRangeException: | Threw OOB as program only set up for 4 keepers. When checking for spaces available when spaces are -1, a message is displayed. |

Screen Dumps

Text

Description automatically generated

Figure Main Menu

A picture containing application

Description automatically generated

Figure Mammal Options

Text

Description automatically generated

Figure Adding Mammal Details

Text

Description automatically generated

Figure Printing Animal Details

Graphical user interface, text, application

Description automatically generated with medium confidence

Figure Reptile Options

Text

Description automatically generated

Figure Adding Reptile Details

Text

Description automatically generated

Figure Printing Reptile Details

Text

Description automatically generated

Figure Allocating Keeper to Cage

Text

Description automatically generated

Figure No Spaces Available in Cage

Text

Description automatically generated

Figure Keeper Has Multiple Cages

Text

Description automatically generated

Figure Keeper has Too Many Cages