Bunnies and Interfaces

```java
public interface Bunny
{
    public void moveBunny(int direction);
    public void displayMove(int direction);
}
```

```java
public interface NamedBunny
{
    public String getName();
    public void displayInfo();
}
```

```java
public interface BunnyHerd
{
    public void printHerd();
    public void addBunny(int x, int y, int numCarrots, String name);
    public void changeHerdName(String newHerdName);
} 
```

Example of BunnyHerd:

```java
BunnyHerd herd = new BunnyHerd();
herd.addBunny(10, 20, 5, "Bunnie 1");
herd.addBunny(30, 40, 10, "Bunnie 2");
herd.printHerd();
```

```java
public class BunnyHerd implements BunnyHerd
{
    private static Bunny[] herd = new Bunny[10];
    private int nextAvailableIndex = 0;

    public void addBunny(int x, int y, int numCarrots, String name)
    {
        if (nextAvailableIndex == herd.length)
        {
            return;
        }

        herd[nextAvailableIndex] = new LittleBunny(x, y, numCarrots, name);
        nextAvailableIndex++;
    }

    public void printHerd()
    {
        for (int i = 0; i < nextAvailableIndex; i++)
        {
            System.out.println(herd[i].getName());
        }
    }
}
```

Question 5 (10 marks)
The world desperately needs better bunny management software, so please help by writing a BunnyHerd class. A BunnyHerd object holds an array of Bunny objects. Your BunnyHerd class definition should include the following four methods:

- constructor: Expects parameters, the X- and Y-coordinates of the bunny, the number of carrots, and the name. This constructor creates a new Bunny object and stores the reference to the object in the next available location in the BunnyHerd object.
- addBunny(x, y, numCarrots, name): Expects four parameters, the X and Y coordinates of the bunny, the number of carrots, and the name. This method creates a new Bunny object and stores the reference to the object in the next available location in the BunnyHerd object.
- printHerd(): This method uses the toString() method of the Bunny object to print information about every Bunny in the herd.
- changeHerdName(newHerdName): Changes the name of the herd from the current name to the new herd name. The method does not change the pointer to the next available location in the BunnyHerd object.

Consider the following program:

```java
public class ParamTest1
{
    public static void main (String[] args)
    {
        int x = 5;
        int y = 3;
        System.out.println("x = " + x);
        System.out.println("y = " + y);
        y = y - 2;
        System.out.println("y is now " + y);
        x = x * 2;
        System.out.println("method1: x is now " + x);
        method1(10);
        System.out.println("method1: x is " + x);
        x = x * x;
        System.out.println("method1: x is now " + x);
        System.out.println("main: number is " + x);
    }

    public static void method1(int number)
    {
        System.out.println("method1: x is now " + number);
        number = number + 1;
        System.out.println("method1: x is now " + number);
    }
}
```

The flow of control:

1. main
   - x = 5
   - y = 3
   - x = 10
   - y = 1
   - x = 100
   - number = 10
   - x = 100
   - number = 11
   - number = 12

2. method1
   - number = 10
   - number = 11
   - number = 12
Consider the following program:

```java
public class ParamTest1 {
  public static void main (String[] args) {
    int number = 4;
    System.out.println("main: number is "+ number);
    method1(number);
    System.out.println("main: number is now "+ number);
  }
}
```

What's the flow of control?

Because when the value in the `int` variable number is passed to method1, what really happens is that a copy of the value (4) in number is assigned to the parameter `x`.

Why not 16?

Because when the value in the `int` variable number is passed to method1, what really happens is that a copy of the value (4) in number is assigned to the parameter `x`.
Parameter Passing

Will this program behave differently? Why or why not?

public class ParamTest2{
  public static void method1(int number) {
    System.out.println("method1: number is now "+ number);
    number = number * number;
    System.out.println("method1: number is now "+ number);
  }
}

What's printed?

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int x[] = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's in foo? Is it the int[] array object? No, it's the reference, or pointer, to the object.

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's printed?

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's in foo? Is it the int[] array object? No, it's the reference, or pointer, to the object. A copy of that reference is passed to method1 and assigned to x.

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

Why not 4?

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's in foo? Is it the int[] array object? No, it's the reference, or pointer, to the object. A copy of that reference is passed to method1 and assigned to x.

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's printed?

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's in foo? Is it the int[] array object? No, it's the reference, or pointer, to the object. A copy of that reference is passed to method1 and assigned to x.

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's in foo? Is it the int[] array object? No, it's the reference, or pointer, to the object. A copy of that reference is passed to method1 and assigned to x.

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's in foo? Is it the int[] array object? No, it's the reference, or pointer, to the object. A copy of that reference is passed to method1 and assigned to x.

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's in foo? Is it the int[] array object? No, it's the reference, or pointer, to the object. A copy of that reference is passed to method1 and assigned to x.

Parameter Passing

Now consider this program.

public class Ptest{
  public static void main(String[] args) {
    int[] x = new int[1];
    x[0] = 4;
    System.out.println("main: number is now: " + number);
    number = number * number;
    System.out.println("method1: x is now: " + x[0]);
  }
}

What's in foo? Is it the int[] array object? No, it's the reference, or pointer, to the object. A copy of that reference is passed to method1 and assigned to x.
Parameter Passing

- Passing primitive types (int, double, boolean) as parameter in Java
  - "pass by value"
  - value in variable is copied
  - copy is passed to method
  - modifying copy of value inside called method
    - modifying a copy of value inside called method has no effect on original value outside called method
  - modifying a copy of value inside called method has no effect on original value outside called method

- Passing object as parameter in Java
  - "pass by reference"
  - objects could be huge, so do not pass copies around
  - pass copy of the object reference
  - object reference aka pointer
  - modifying object pointed to by reference inside calling method does affect object pointed to by reference outside calling method
    - both references point to same object

Variable Types

- Static variables
  - declared within class
  - associated with class, not instance
  - static fields belong to whole class
  - nonstatic fields belong to instantiated object
  - Static methods can only use static fields
    - nonstatic methods can use either nonstatic or static fields
  - Static fields can only use static fields

- Instance variables
  - declared within class
  - associated with instance
  - access controlled throughout object, lifetime of object
  - Instance variables
  - declared within class
  - associated with instance
  - access controlled throughout object, lifetime of object

- Local variables
  - declared within method
  - access controlled throughout method, lifetime of method
  - Local variables
  - declared within method
  - access controlled throughout method, lifetime of method

- Parameters
  - declared in parameter list of method
  - access controlled throughout method, lifetime of method

Parameter Passing Pictures

- object as parameter:
  - copy of pointer made
  - copy of value

Review: Static Fields/Methods

- Static fields belong to whole class
  - nonstatic fields belong to instantiated object
  - Static methods can only use static fields
    - nonstatic methods can use either nonstatic or static fields

Variable Scope

- Scope of a variable (or constant) is that part of a program in which value of that variable can be accessed

Variable Types

- Static instance? Local? Parameters?
  - class: Giraffe
    - getGiraffeCount()
    - setGiraffeCount()
    - setNeckLength()
    - getNeckLength()

Review: Variable Scope

- Scope of a variable (or constant) is that part of a program in which value of that variable can be accessed