Review of Functional Programming

Topical Review Session 01

Announcements

Session 01

- ❏ Structure of these sessions
- ❏ Discord
- ❏ Worksheet

Structure of Java

Syntactic Structure

Python

- Indentation is significant and is used to mark syntactic structures (statement blocks and code within function, class, and method definitions, etc)
- A phrase can be broken and continued on the next line after a comma, or by using the '\' symbol
- The headers of control statements and functions, classes, and method definitions end with a colon (:)

Java

- Indentation is not significant, so all lexical items are separated by zero or more spaces
- Blocks of code in syntactic structures are enclosed in curly braces ({})
- Simple statements end with a semicolon (;)
- Boolean expressions in loops and if statements must be enclosed in parentheses

Spot the Differences

Python

```
stuff = ["Hello, World!", "Hi there, Everyone!", 6]
for i in stuff:
    print(i)
```

Java

```
public class Test {
    public static void main(String args[]) {
        String[] array = {"Hello, World", "Hi there, Everyone", "6"};
        for (String i : array) {
            System.out.println(i);
        }
    }
}
```

Variables

Box Diagrams

```
public class Frog {
	public String name;
	public String sound;
	public Frog(String name, String sound) {
		this.name = name;
		this.sound = sound;
	}

	public static void main(String[] args) {
		Frog a = new Frog(“Froggy”, “Ribbit”);
		Frog b = new Frog(“Frogger”, “Croak”);
	}
}
```

Box Diagrams

```
public class Frog {
      public String name;
      public String sound;
      public Frog(String name, String sound) {
            this.name = name;
            this.sound = sound;
      }

      public static void main(String[] args) {
            Frog a = new Frog("Froggy", "Ribbit");
            Frog b = new Frog("Frogger", "Croak");
      }
}
```

Box Diagrams

```
public class Frog {
      public String name;
      public String sound;
      public Frog(String name, String sound) {
            this.name = name;
            this.sound = sound;
      }

      public static void main(String[] args) {
            Frog a = new Frog("Froggy", "Ribbit");
            Frog b = new Frog("Frogger", "Croak");
      }
}
```

Static vs. Instance Variables

- Both static and instance variables are declared and instantiated within a class outside any method or constructor
- The difference between the syntax is that static variables will have the keyword `static` in front of it while instance variables do not
- **Static variables** are the class' variable, they are not different amongst different instances
  - Static variables are accessed using *ClassName.VariableName*
- **Instance variables** are unique to each instance of the class
  - Instance variables are accessed using *objectName.VariableName*
    - Essentially through the instances of objects that are created

```
public class Frog {
      public String name;
      public Frog(String name) {
            this.name = name;
      }
}
```

```
public static void main(String[] args) {
      Frog a = new Frog("Steve");
      String steveName = a.name;
}
```

Box Diagrams

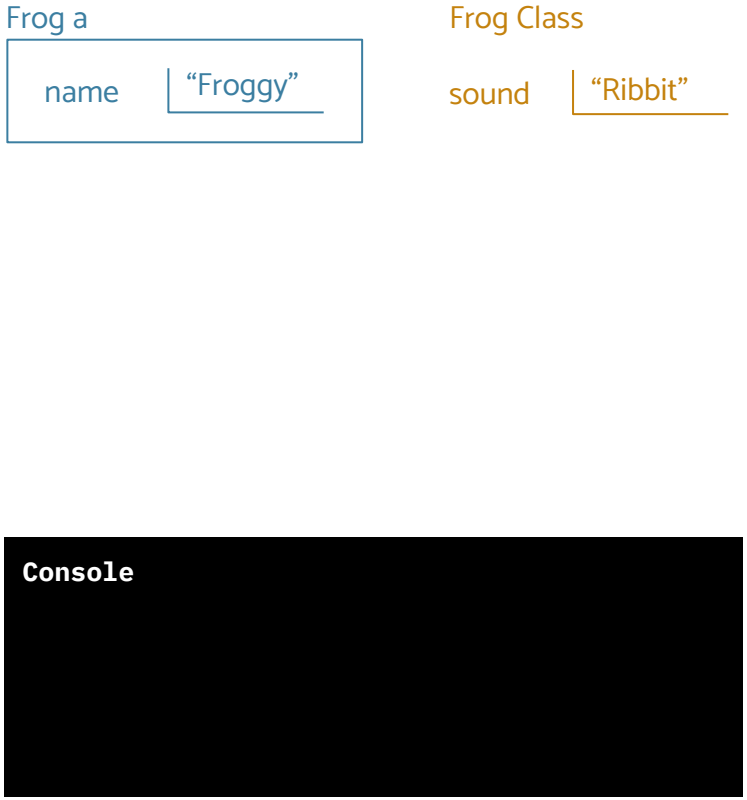
```
public class Frog {
      public String name;
      public static String sound;
      public Frog(String name, String sound) {
            this.name = name;
            this.sound = sound;
      }

      public static void main(String[] args) {
            Frog a = new Frog(“Froggy”, “Ribbit”);
            System.out.println(a.sound);
            Frog b = new Frog(“Frogger”, “Croak”);
            System.out.println(a.sound);
            System.out.println(b.sound);
      }
}
```

Box Diagrams

```
public class Frog {
      public String name;
      public static String sound;
      public Frog(String name, String sound) {
            this.name = name;
            this.sound = sound;
      }

      public static void main(String[] args) {
            Frog a = new Frog("Froggy", "Ribbit");
            System.out.println(a.sound);
            Frog b = new Frog("Frogger", "Croak");
            System.out.println(a.sound);
            System.out.println(b.sound);
      }
}
```



Box Diagrams

```
public class Frog {
      public String name;
      public static String sound;
      public Frog(String name, String sound) {
            this.name = name;
            this.sound = sound;
      }

      public static void main(String[] args) {
            Frog a = new Frog("Froggy", "Ribbit");
            System.out.println(a.sound);
            Frog b = new Frog("Frogger", "Croak");
            System.out.println(a.sound);
            System.out.println(b.sound);
      }
}
```

Box Diagrams

```
public class Frog {
      public String name;
      public static String sound;
      public Frog(String name, String sound) {
            this.name = name;
            this.sound = sound;
      }

      public static void main(String[] args) {
            Frog a = new Frog("Froggy", "Ribbit");
            System.out.println(a.sound);
            Frog b = new Frog("Frogger", "Croak");
            System.out.println(a.sound);
            System.out.println(b.sound);
      }
}
```

Console

```
Ribbit
```

Box Diagrams

```
public class Frog {
	public String name;
	public static String sound;
	public Frog(String name, String sound) {
		this.name = name;
		this.sound = sound;
	}

	public static void main(String[] args) {
		Frog a = new Frog("Froggy", "Ribbit");
		System.out.println(a.sound);
		Frog b = new Frog("Frogger", "Croak");
		System.out.println(a.sound);
		System.out.println(b.sound);
	}
}
```

Console

```
Ribbit
Croak
```

Box Diagrams

```
public class Frog {
	public String name;
	public static String sound;
	public Frog(String name, String sound) {
		this.name = name;
		this.sound = sound;
	}

	public static void main(String[] args) {
		Frog a = new Frog("Froggy", "Ribbit");
		System.out.println(a.sound);
		Frog b = new Frog("Frogger", "Croak");
		System.out.println(a.sound);
	  → System.out.println(b.sound);
	}
}
```

Console

```
Ribbit
Croak
Croak
```

Loops & Recursion

Loops

Python

```
i = 0
while i < 5:
    print(i)
    i = i + 1
```

Java

```
???
```

Loops

Python

```
i = 0
while i < 5:
    print(i)
    i = i + 1
```

Java

```
???
```

Console

```
0
1
2
3
4
```

Loops

Python

```
i = 0
while i < 5:
    print(i)
    i = i + 1
```

Java

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

OR

```
int i = 0;
while (i < 5) {
    System.out.println(i)
    i++;
}
```

Console

```
0
1
2
3
4
```

Loop Structure

```
for (exp 1; exp 2; exp 3) {
     //Code
}
```

OR

```
exp 1;
while (exp 2) {
     //Code
     exp 3;
}
```

Expression 1 : Initialize the variables (int i = 0)

Expression 2 : The condition (i < 5)

Expression 3 : The increment (i++)

For-Each Loops

Python

```
numList = [0, 1, 2, 3, 4]
for i in numList:
    print(i)
```

Java

```
???
```

For-Each Loops

Python

```
numList = [0, 1, 2, 3, 4]
for i in numList:
    print(i)
```

Java

```
???
```

Console

```
0
1
2
3
4
```

For-Each Loops

Python

```
numList = [0, 1, 2, 3, 4]
for i in numList:
    print(i)
```

Java

```
int[] numList = {0, 1, 2, 3, 4}
for (int i : numList) {
    System.out.println(i);
}
```

Console

```
0
1
2
3
4
```

Recursion

Steps

1. **Base Case** : Simple cases which can be solved without recursion
2. Recursive Step : Cases where the function calls itself
3. Leap of Faith :
    a. Make sure you're covering all base cases
    b. Assume the recursive function call will return the correct value
    c. Given this assumption, check if your function returns the correct value

```
public int fib (int n) {

    // Base Case

    // Recursive Step

}
```

Recursion

Steps

1. Base Case : Simple cases which can be solved without recursion
2. **Recursive Step** : Cases where the function calls itself
3. Leap of Faith :
    a. Make sure you're covering all base cases
    b. Assume the recursive function call will return the correct value
    c. Given this assumption, check if your function returns the correct value

```
public int fib (int n) {

    if (n == 0 || n == 1) {
        return n;
    }

    // Recursive Step

}
```

Recursion

Steps

1. Base Case : Simple cases which can be solved without recursion
2. Recursive Step : Cases where the function calls itself
3. **Leap of Faith** :
    a. Make sure you're covering all base cases
    b. Assume the recursive function call will return the correct value
    c. Given this assumption, check if your function returns the correct value

```
public int fib (int n) {

    if (n == 0 || n == 1) {
        return n;
    }

    return fib(n - 1) + fib(n - 2);

}
```