Outline

- Scope
- Garbage Collection

Scope

- Scope is about two things:
  - Access of variables.
  - Lifespan of variables.
- Variables are like mailboxes:
  - A place to store things.
  - At some point the name is removed and another name is put on the box.

Scope Example: Lifespan of Inner Block Variable

```
public static void main(String[] args) {
  if (3 < 5) {
    int x = 5; // <-- Variable created
    x = x + 3; // <-- Variable in scope
  } // <-- Variable lost from view
  int x = 37; // Completely different variable
}
```

Scope Example: Inner Block Access

```
public class Example {
  public static void main(String[] args) {
    int x = 5;                          // x->[ 5 ]
    for (int i = 0; i < 3; i++) {       // |
      System.out.println(x);            // | Accessible
    }                                   // V
  }                                     // variable lost
}
```

- `x` is accessible inside the inner block.
- `{` and `}` define the block.

Scope Example: Class Field

```
public class Tree {
  private int x = 5;          // x->[ 5 ]
                              // |
  public Tree() {             // V
    x = 6;                    // x [ 6 ] available
  }                           // |
                              // |
  public void printX() {      // V
    System.out.println(x); // x "6" still available
  }                           // |
}                             // V
```

```
Tree tree = new Tree();     // x is hidden under the tree
```

- `x` is accessible from any method in class `Tree`.
- An `x` lives as long as the `Tree` object does.

Scope Question

```
public class Question {
  public static void main(String[] args) {
    int x = 5; // Where else can x be found?
    while (x > 4) {
      // 1. x?
    }
    // 2. x?
  }

  public static printX() {
    // 3. x?
  }
}
```

- Where is `x` accessible?

Scope Question 2

```
public class Question2 {
  public static void main(String[] args) {
    if (5 > 4) {
      int x = 5;
      // 1. x?
    }
    // 2. x?
  }

  public static printX() {
    // 3. x?
  }
}
```

Scope Question 3

```
public class Question3 {
  public int x = 3;
  // 1. x?
}
```

```
public class Runner {
  public static void main(String[] args) {
    Question3 q3 = new Question3();
    // 2. x?
    printX(q3);
  }

  public static void printX(Question3 thing) {
    // 3. x?
    System.out.println(thing.x);
  }
}
```

Garbage Collection

- Memory:
  - All computers have limited memory.
  - Some programs fill up all the memory.
- Remember the following slides when you get:
  - `java.lang.OutOfMemoryError`

Garbage Collection Process

- Runs separate from main program
- Checks for objects no longer in use.
- Reclaims memory when Java decides to.

Garbage Collection: Example

```
                                        // Heap
Monster m1 = new Monster("Taco");       // ----
// m1[ c375 ] ------------------------> // [ Monster A ]
Monster m2 = new Monster("Bike");
// m2[ c382 ] ------------------------> // [ Monster B ]
```

- Object creation:
  - Local variable stores a memory address.
  - All fields, functions stored in section of memory called "Heap".

Garbage Collection: Example

```
                                        // Heap
m1 = new Monster("Truck");              // ----
//                                  X-> // [ Monster A ]
// m1[ c398 ] ------------------------> // [ Monster C ]
// m2[ c382 ] ------------------------> // [ Monster B ]
```

- `Monster A` [`new Monster("Taco")`] has been lost to our program.
- Garbage Collector will come along and see nothing points to `Monster A` and reclaim that space.

Garbage Collection: Example

```
                                        // Heap
m1 = m2;                                // ----
//                                  X-> // [ Monster A ]
// m1[ c382 ] --\                   X-> // [ Monster C ]
// m2[ c382 ] --t-----------------> // [ Monster B ]
```

- Reassign `m1` to `Monster B`.
- Now there are two lost Monsters A and C.
- They will be cleaned up by Garbage Collection.