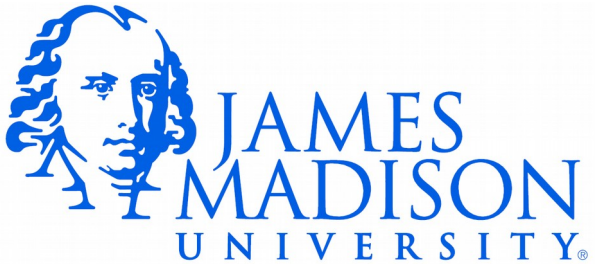
CS139 – References, Testing



Reference Variables vs Primitive Variables

- Variable – Named location in memory.
- Primitive variable – The value is actually stored at that location.

```
int number;
```

- Reference variable – contains the *address* of an object.

```
String name;

Scanner input;
```

Objects Tie Together Data and Methods

- *Not* possible to call methods on primitive variables:

```
int count = 3;
count.someMethod(); // NOOOO!
```

- We *can* call methods on objects:
  - `Scanner input = new Scanner(System.in);`
  - `count = scanner.nextInt();`

Casting Primitive Types

- Cast – Ask Java to perform a type conversion that it wouldn't otherwise do.

```
byte small  = 10;
int bigger  = 10;
double huge = 10;

small  = bigger;  // Won't compile!
small  = huge;    // Won't compile!
bigger = huge;    // Won't compile!

small  = (byte)bigger; // OK!
small  = (byte)huge;   // OK!
bigger = (int)huge;    // OK!
```

Writing and Testing Methods

- Scenario: We are developing an application for a shipping company to help them load trucks more efficiently:
- Repeat the following:
  - Query user for the number of packages that need to be shipped and their weight.
  - Select a truck that has enough capacity for the load. Try to fill one truck as much as possible before loading another.
  - Report when a truck is ready to depart.

Development Process

- Steps:
  - Break the problem down into manageable, testable parts
  - Design methods for addressing each part
  - Stub out the methods
  - Develop and test the methods in isolation
  - Combine the methods to solve the overall problem

Shipping Application Task Decomposition

- Obtain user input:
  - What trucks are available
  - # packages that need to be shipped along with their weight
- Perform computations
  - Select the next truck
    - Compute how many packages each truck can hold
    - How much space remains after adding the load
    - Compare across all trucks
- Generate output...

Stubbed Method

```
/**
 * Calculate the number of additional packages we can add to a partially
 * loaded truck without exceeding a weight limit.
 *
 * @param totalCapacity The total capacity of the truck in pounds
 * @param numLoaded The number of packages that have already been loaded
 * @param packageWeight The weight of each individual package
 * @return the number of additional packages that can be loaded without
 *         exceeding the capacity
 */
public static int packageCapacity(int totalCapacity, int numLoaded,
                                  int packageWeight) {
   return -1;
}
```

Developing Test Cases

totalCapacity	numLoaded	packageWeight	Expected output
0	0	10	
1000	0	250	
1000	0	501	
2000	2	1000	
2000	1	1000	
2000	1	1001	
2000	1	501	

Developing Test Cases

totalCapacity	numLoaded	packageWeight	Expected output
0	0	10	0
1000	0	250	4
1000	0	501	1
2000	2	1000	0
2000	1	1000	1
2000	1	1001	0
2000	1	501	2

Test Driver...

```
public class ShippingDriver {

   public static void main(String[] args) {

      testPackageCapacity();
   }

   public static void testPackageCapacity() {
      int capacity;

      capacity = Shipping.packageCapacity(0, 0, 10);
      System.out.println("Expected: 0" + " Actual: " + capacity);

      capacity = Shipping.packageCapacity(1000, 0, 250);
      System.out.println("Expected: 4" + " Actual: " + capacity);

      capacity = Shipping.packageCapacity(1000, 0, 501);
      System.out.println("Expected: 1" + " Actual: " + capacity);

      capacity = Shipping.packageCapacity(2000, 2, 1000);
      System.out.println("Expected: 0" + " Actual: " + capacity);

      capacity = Shipping.packageCapacity(2000, 1, 1000);
      System.out.println("Expected: 1" + " Actual: " + capacity);

      capacity = Shipping.packageCapacity(2000, 1, 1001);
      System.out.println("Expected: 0" + " Actual: " + capacity);

      capacity = Shipping.packageCapacity(2000, 1, 501);
      System.out.println("Expected: 2" + " Actual: " + capacity);

   }
}
```

We will see less cumbersome way of doing this later in the semester...