Introduction to Java (All the Basic Stuff)

Georgia Tech

Learning Objectives

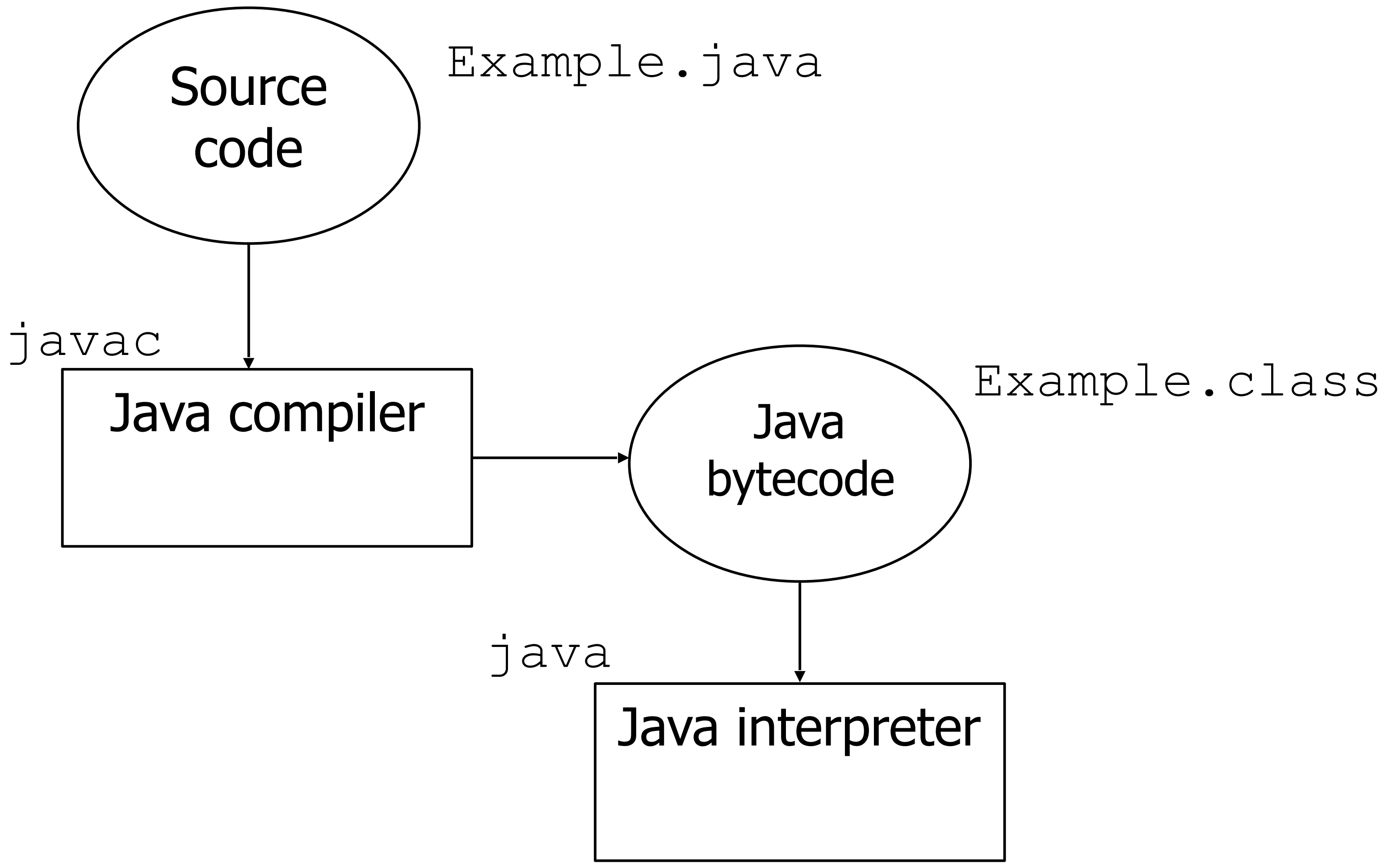
- Java's edit-compile-run loop
- Basics of object-oriented programming
- Classes
  - objects, instantiation, methods
- Primitive types
- Math expressions
- Output, input
- Strings

Programs

- Python – interpreted
  - Interweaves translation and execution
- C – compiled
  - High-level source code
  - Assembly code
  - Machine language code

- Java is a kind of hybrid

Overview

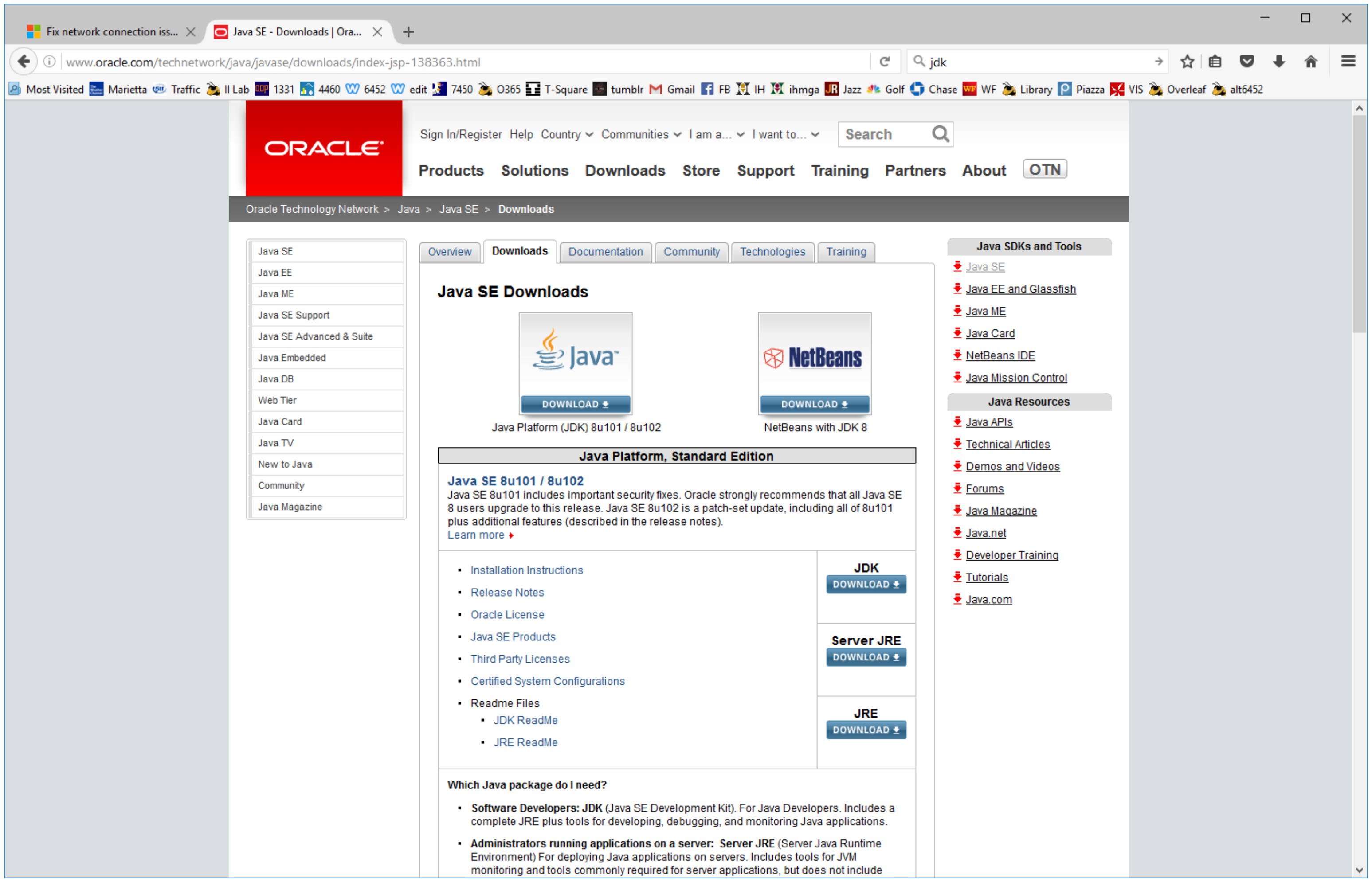

Source code
Example.java
javac
Java compiler
Java bytecode
Example.class
java
Java interpreter

Java Downloads

- JRE – Java runtime environment
- JDK – Java SE Development Kit

- Differences

Getting Java



Java SE Downloads

Java Platform (JDK) 8u101 / 8u102

NetBeans with JDK 8

Java Platform, Standard Edition

Java SE 8u101 / 8u102

Java SE 8u101 includes important security fixes. Oracle strongly recommends that all Java SE 8 users upgrade to this release. Java SE 8u102 is a patch-set update, including all of 8u101 plus additional features (described in the release notes).

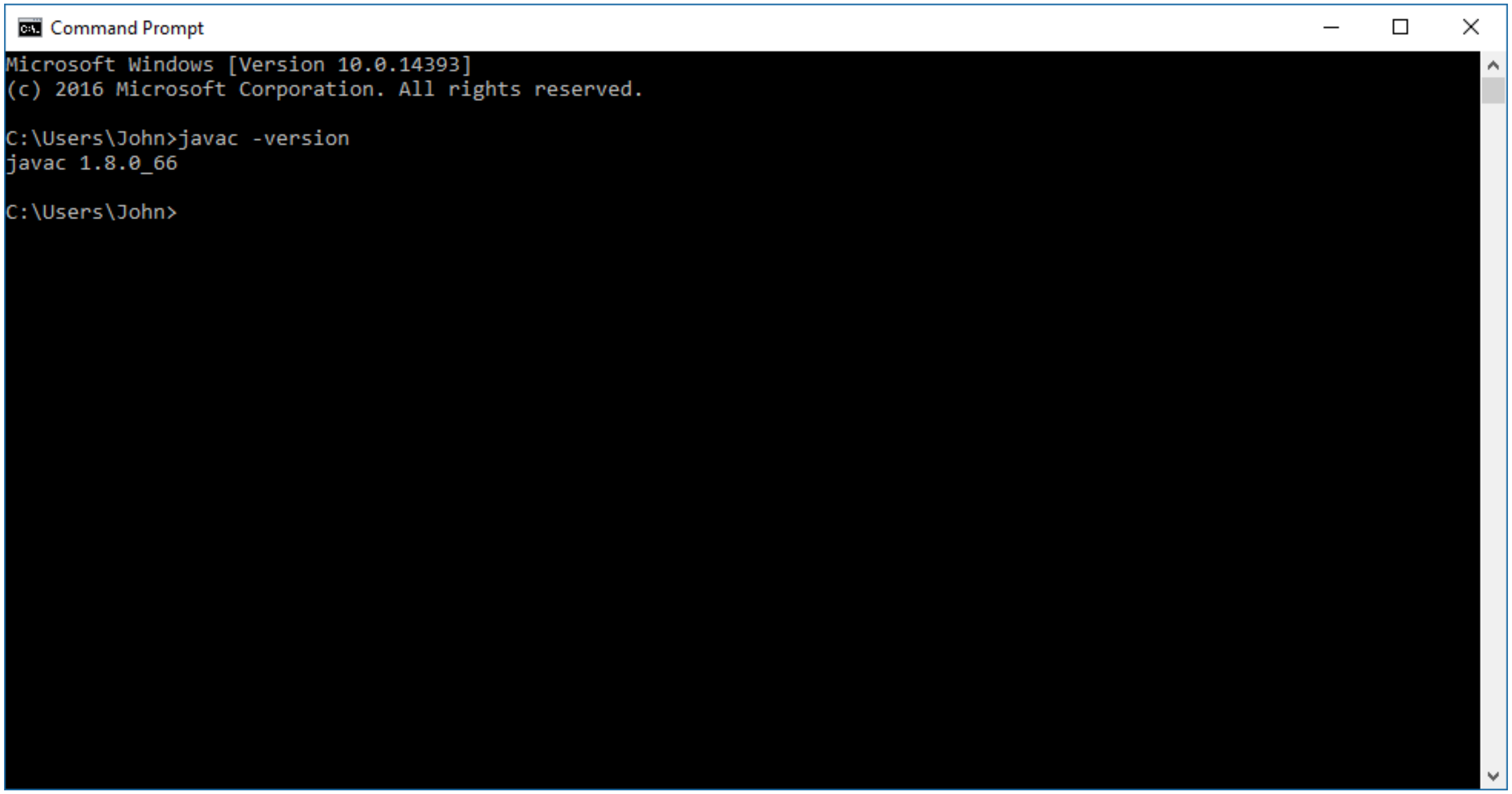
Learn more

- Installation Instructions
- Release Notes
- Oracle License
- Java SE Products
- Third Party Licenses
- Certified System Configurations
- Readme Files
  - JDK ReadMe
  - JRE ReadMe

Which Java package do I need?

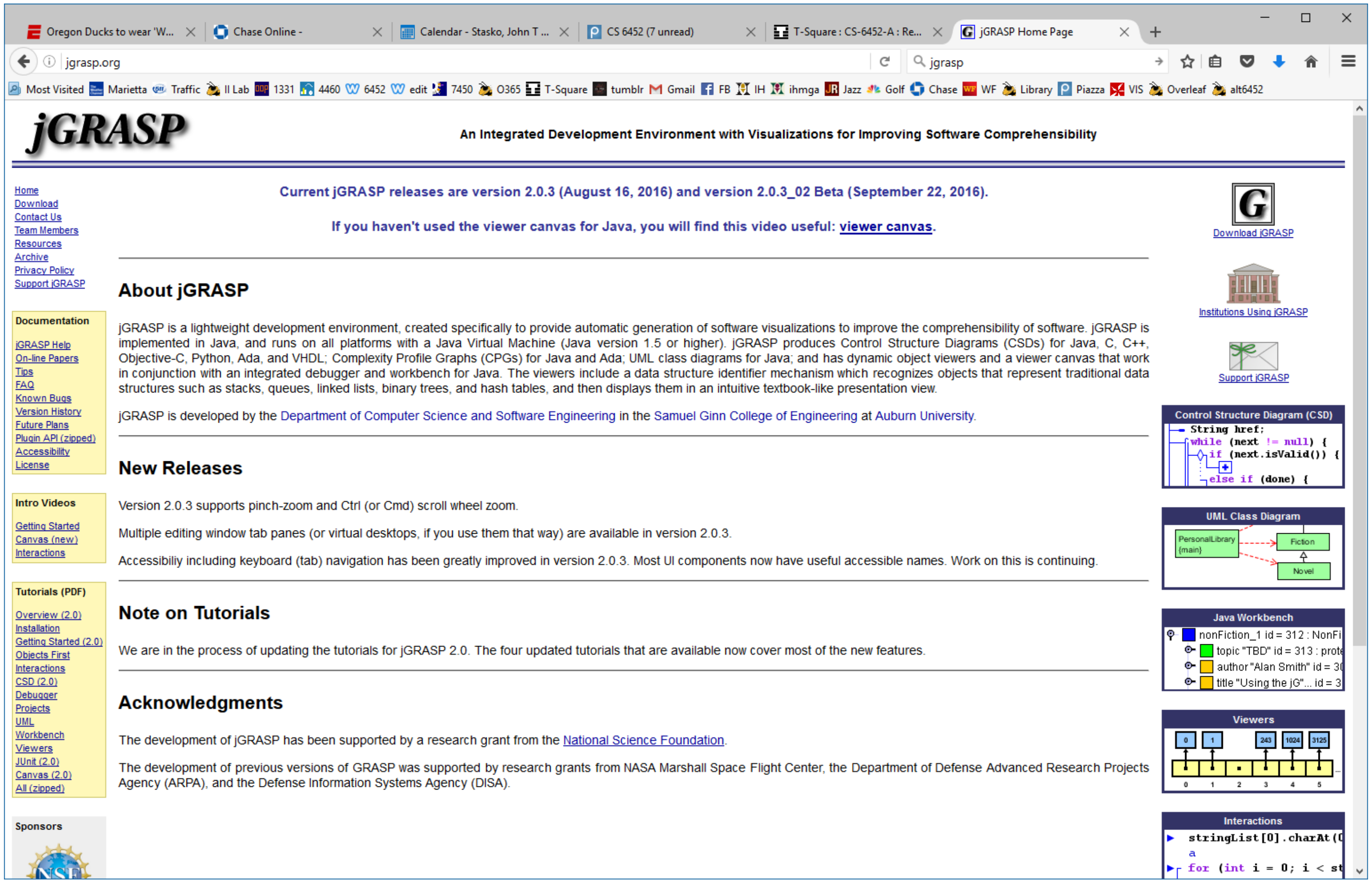
- **Software Developers: JDK** (Java SE Development Kit). For Java Developers. Includes a complete JRE plus tools for developing, debugging, and monitoring Java applications.
- **Administrators running applications on a server: Server JRE** (Server Java Runtime Environment) For deploying Java applications on servers. Includes tools for JVM monitoring and tools commonly required for server applications, but does not include

Test It



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\John>javac -version
javac 1.8.0_66

C:\Users\John>
```

Gentle IDE



jGRASP

An Integrated Development Environment with Visualizations for Improving Software Comprehensibility

Home
Download
Contact Us
Team Members
Resources
Archive
Privacy Policy
Support jGRASP

Documentation

jGRASP Help
On-line Papers
Tips
FAQ
Known Bugs
Version History
Future Plans
Plugin API (zipped)
Accessibility
License

Intro Videos

Getting Started
Canvas (new)
Interactions

Tutorials (PDF)

Overview (2.0)
Installation
Getting Started (2.0)
Objects First
Interactions
CSD (2.0)
Debugger
Projects
UML
Workbench
Viewers
JUnit (2.0)
Canvas (2.0)
All (zipped)

Sponsors

Current jGRASP releases are version 2.0.3 (August 16, 2016) and version 2.0.3_02 Beta (September 22, 2016).

If you haven't used the viewer canvas for Java, you will find this video useful: viewer canvas.

About jGRASP

jGRASP is a lightweight development environment, created specifically to provide automatic generation of software visualizations to improve the comprehensibility of software. jGRASP is implemented in Java, and runs on all platforms with a Java Virtual Machine (Java version 1.5 or higher). jGRASP produces Control Structure Diagrams (CSDs) for Java, C, C++, Objective-C, Python, Ada, and VHDL; Complexity Profile Graphs (CPGs) for Java and Ada; UML class diagrams for Java; and has dynamic object viewers and a viewer canvas that work in conjunction with an integrated debugger and workbench for Java. The viewers include a data structure identifier mechanism which recognizes objects that represent traditional data structures such as stacks, queues, linked lists, binary trees, and hash tables, and then displays them in an intuitive textbook-like presentation view.

jGRASP is developed by the Department of Computer Science and Software Engineering in the Samuel Ginn College of Engineering at Auburn University.

New Releases

Version 2.0.3 supports pinch-zoom and Ctrl (or Cmd) scroll wheel zoom.

Multiple editing window tab panes (or virtual desktops, if you use them that way) are available in version 2.0.3.

Accessibiliy including keyboard (tab) navigation has been greatly improved in version 2.0.3. Most UI components now have useful accessible names. Work on this is continuing.

Note on Tutorials

We are in the process of updating the tutorials for jGRASP 2.0. The four updated tutorials that are available now cover most of the new features.

Acknowledgments

The development of jGRASP has been supported by a research grant from the National Science Foundation.

The development of previous versions of GRASP was supported by research grants from NASA Marshall Space Flight Center, the Department of Defense Advanced Research Projects Agency (ARPA), and the Defense Information Systems Agency (DISA).

Download jGRASP

Institutions Using jGRASP

Support jGRASP

Control Structure Diagram (CSD)

```
String href;
while (next != null) {
  if (next.isValid()) {

  else if (done) {
```

UML Class Diagram

Java Workbench

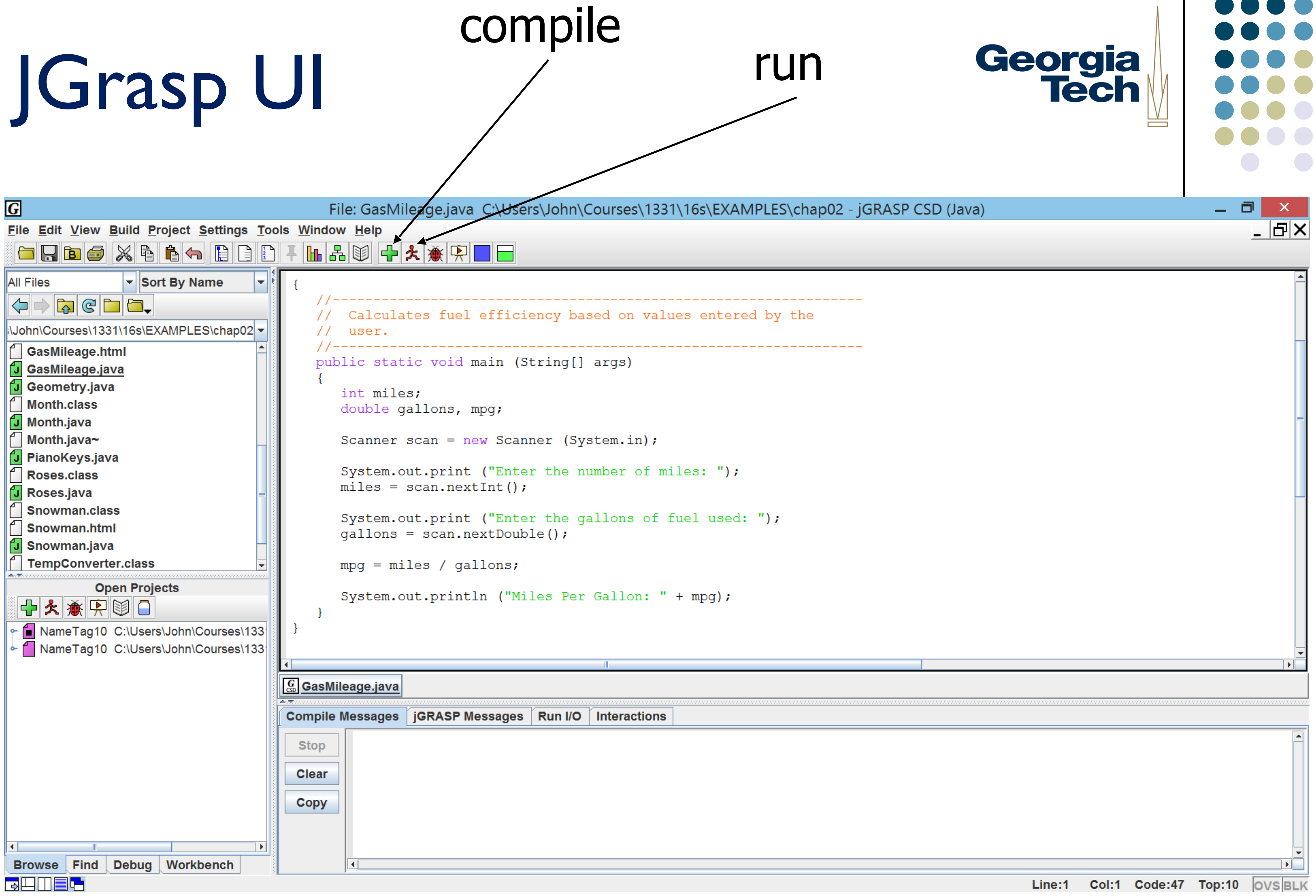
```
nonFiction_1 id = 312 : NonFi
  topic "TBD" id = 313 : prote
  author "Alan Smith" id = 30
  title "Using the jG"... id = 3
```

Viewers

Interactions

```
stringList[0].charAt(0
a
for (int i = 0; i < st
```

JGrasp UI


compile
run
File: GasMileage.java C:\Users\John\Courses\1331\16s\EXAMPLES\chap02 - jGRASP CSD (Java)
File Edit View Build Project Settings Tools Window Help
All Files
Sort By Name
\John\Courses\1331\16s\EXAMPLES\chap02
GasMileage.html
GasMileage.java
Geometry.java
Month.class
Month.java
Month.java~
PianoKeys.java
Roses.class
Roses.java
Snowman.class
Snowman.html
Snowman.java
TempConverter.class
Open Projects
NameTag10 C:\Users\John\Courses\133
NameTag10 C:\Users\John\Courses\133
Browse Find Debug Workbench
GasMileage.java
Compile Messages jGRASP Messages Run I/O Interactions
Stop
Clear
Copy
Line:1 Col:1 Code:47 Top:10 OVS BLK

```
{
   //-----------------------------------------------------------------
   //  Calculates fuel efficiency based on values entered by the
   //  user.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int miles;
      double gallons, mpg;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter the number of miles: ");
      miles = scan.nextInt();

      System.out.print ("Enter the gallons of fuel used: ");
      gallons = scan.nextDouble();

      mpg = miles / gallons;

      System.out.println ("Miles Per Gallon: " + mpg);
   }
}
```

Classes & Objects

- Object-oriented programming
- Classes
  - Model or blueprint from which objects are made
- Object
  - State (attributes)
  - Behaviors (methods)

Car analogy

Program

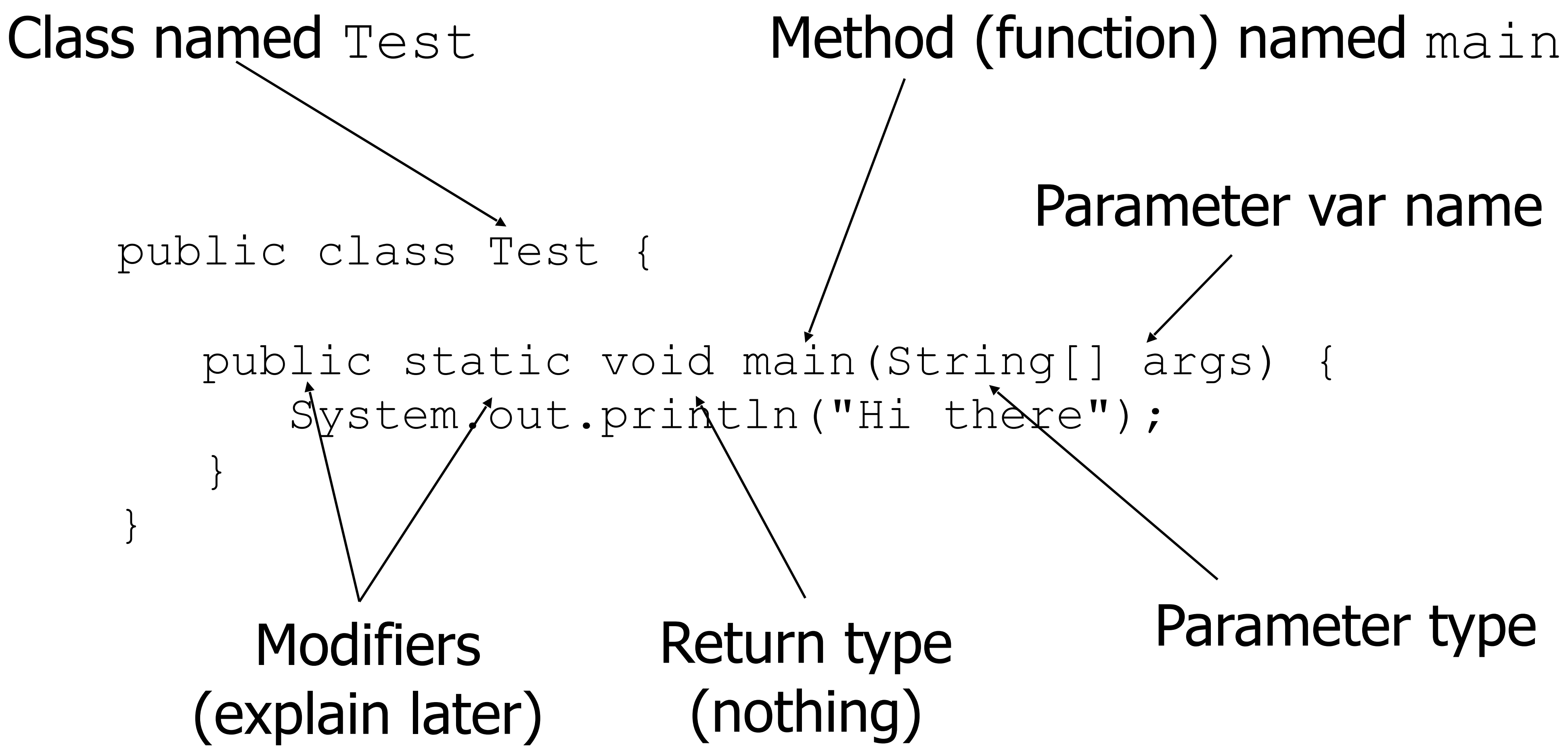
- Made up of classes
  - Each class in its own file (mostly)
- Class named `Tiger` goes in the file `Tiger.java`

Small Example

```
// An example program
public class Test {

   public static void main(String[] args) {
      System.out.println("Hi there");
   }
}

/* A longer comment
    that goes across multiple lines */
```

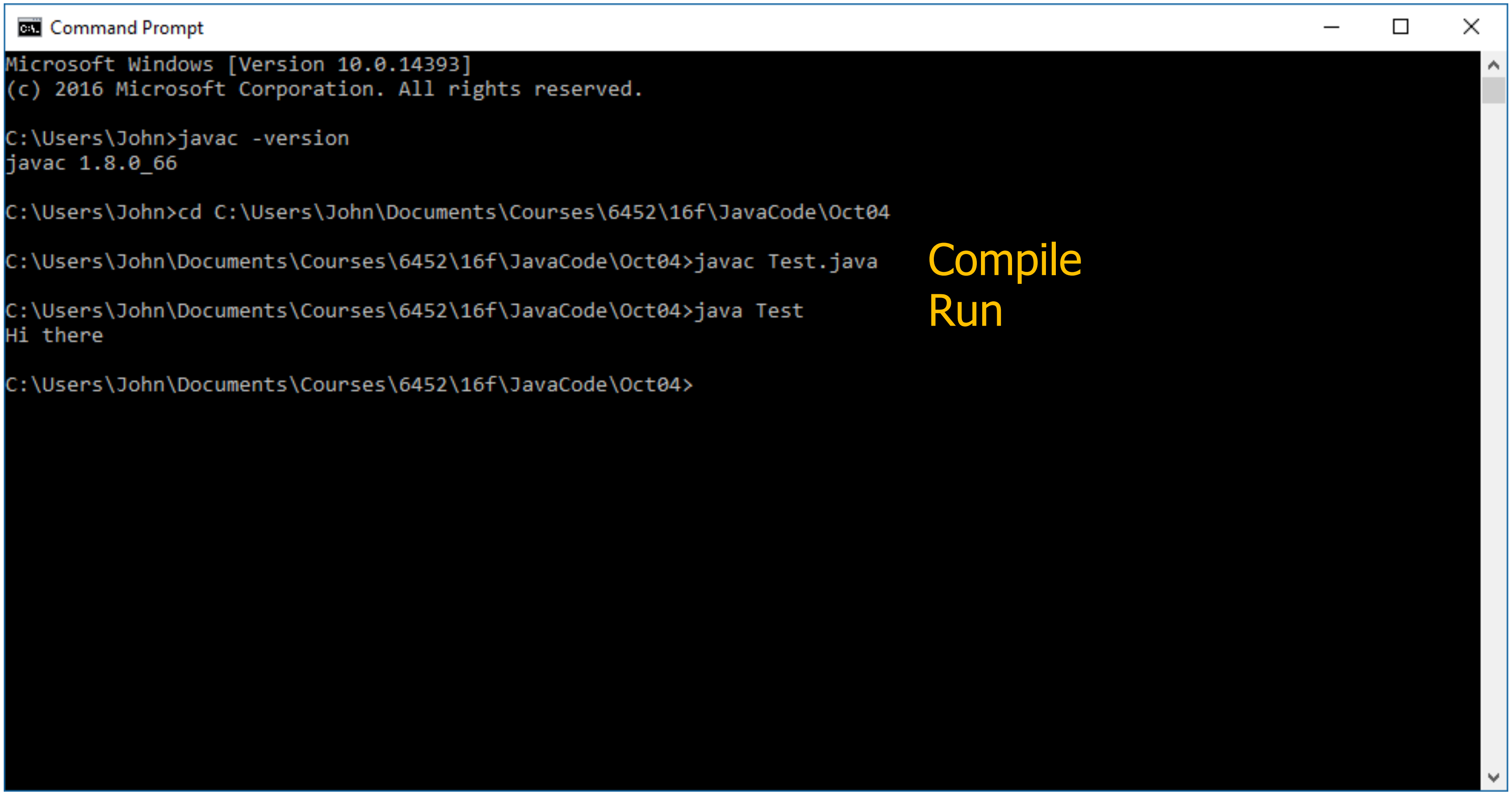
File `Test.java`

What is that?



```
public class Test {

    public static void main(String[] args) {
        System.out.println("Hi there");
    }
}
```

Compile & Run



Command Prompt

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\John>javac -version
javac 1.8.0_66

C:\Users\John>cd C:\Users\John\Documents\Courses\6452\16f\JavaCode\Oct04

C:\Users\John\Documents\Courses\6452\16f\JavaCode\Oct04>javac Test.java

C:\Users\John\Documents\Courses\6452\16f\JavaCode\Oct04>java Test
Hi there

C:\Users\John\Documents\Courses\6452\16f\JavaCode\Oct04>
```

Java Programs

- Strongly typed
  - Variables must be declared
  - Cannot change type later
  - Can only mix compatible types

- All whitespace the same
- Statements separated by ;
- Case sensitive

Entities

- Two types in Java
  - Primitive data
  - Objects
- Java variable holds either primitive value or a reference to an object

Primitive Types

- `int, double, char, boolean`

```
int total = 10;
double f;
char ch = 'P';
boolean done;
```

Others exist too

Math Expressions

```
int i,j,total;
j = 25;
i = 10 * j + 1;

sum = j * i;                    // Error, why?
total = (i + 20) / 46.3;        // Error, why?
```

Output

- `println` – print with a newline
- `print` – print with no newline

```
System.out.println("Way to go!");
System.out.println("The value is "+j+" and I'm out");
```

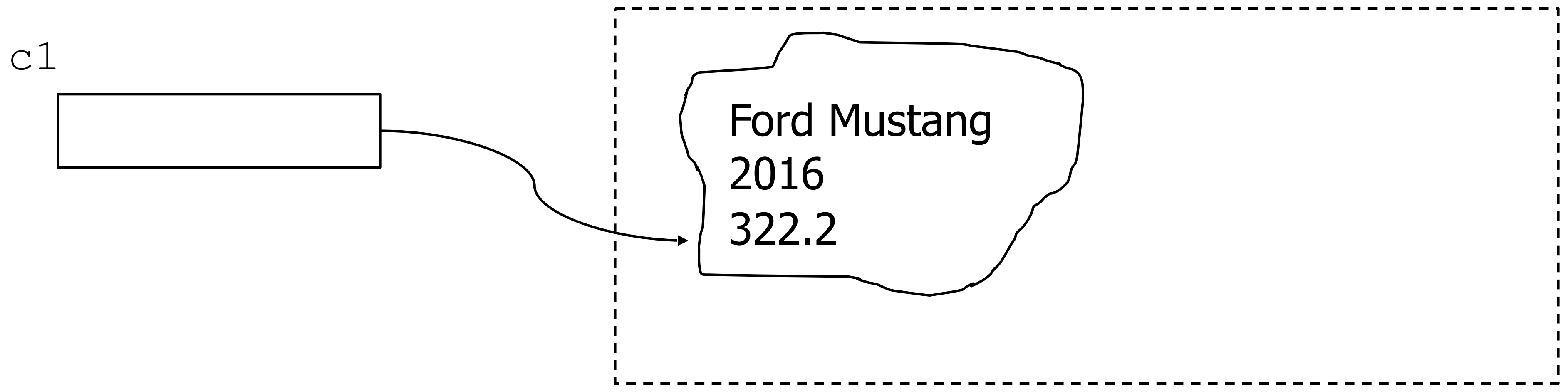
Example Program

```
public class TempConverter
{
   //---------------------------------------------------------------------
   //  Computes the Fahrenheit equivalent of a specific Celsius
   //  value using the formula F = (9/5)C + 32.
   //---------------------------------------------------------------------
   public static void main (String[] args)
   {
      final int BASE = 32;
      final double CONVERSION_FACTOR = 9.0 / 5.0;

      double fahrenheitTemp;
      int celsiusTemp = 24;  // value to convert

      fahrenheitTemp = celsiusTemp * CONVERSION_FACTOR + BASE;

      System.out.println ("Celsius Temperature: " + celsiusTemp);
      System.out.println ("Fahrenheit Equivalent: " + fahrenheitTemp);
   }
}
```

Instantiation

- Creating an instance (object) of a class
- Calls a constructor method to set up object
  - Has exact same name as class
  - Object created by new operator

```
Car c1;
c1 = new Car("Ford Mustang", 2016, 322.2);
```



Access

- We access methods through . operator
- Let's explore provided `String` class

```
String j;
j = new String("Hello");
int len = j.length();
```

String methods

return value	name and parameters
`char`	`charAt(int index)`
`int`	`length()`
`int`	`compareTo(String s)`
`String`	`replace(char oldCh, char newCh)`
`String`	`toLowerCase()`

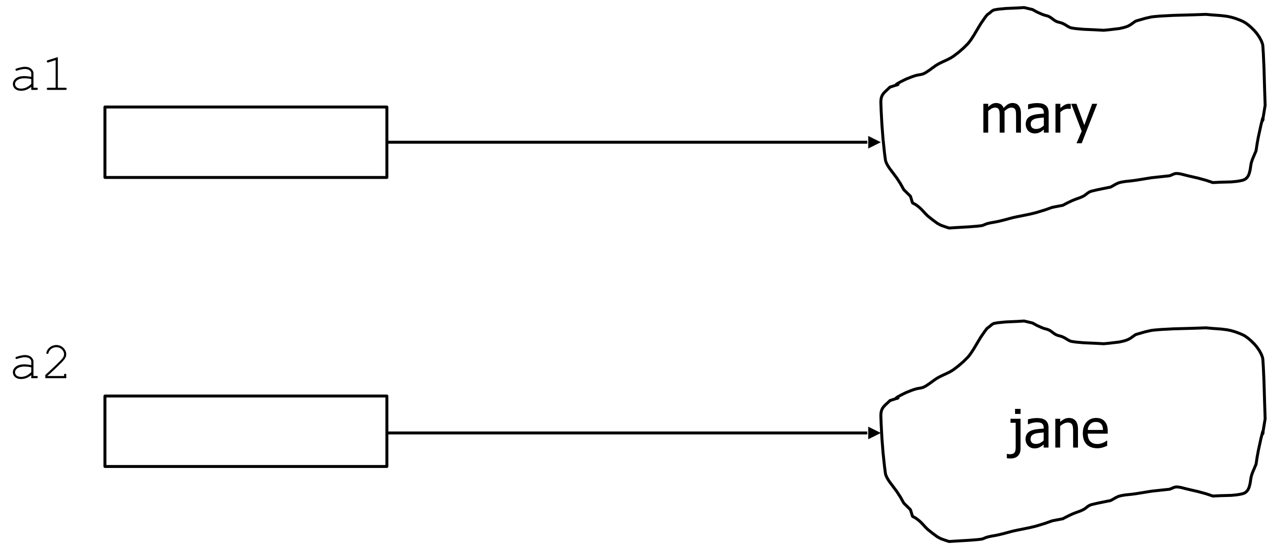
Strings are immutable
(Strings are not arrays/lists of characters)

Example Program

```
public class StringMutation
{
   public static void main (String[] args)
   {
      String phrase = "Change is inevitable";
      String mutation1, mutation2, mutation3, mutation4;

      System.out.println ("Original string: \"" + phrase + "\"");
      System.out.println ("Length of string: " + phrase.length());

      mutation1 = phrase.concat (", except from vending machines.");
      mutation2 = mutation1.toUpperCase();
      mutation3 = mutation2.replace ('E', 'X');
      mutation4 = mutation3.substring (3, 30);

      // Print each mutated string
      System.out.println ("Mutation #1: " + mutation1);
      System.out.println ("Mutation #2: " + mutation2);
      System.out.println ("Mutation #3: " + mutation3);
      System.out.println ("Mutation #4: " + mutation4);

      System.out.println ("Mutated length: " + mutation4.length());
   }
}
```

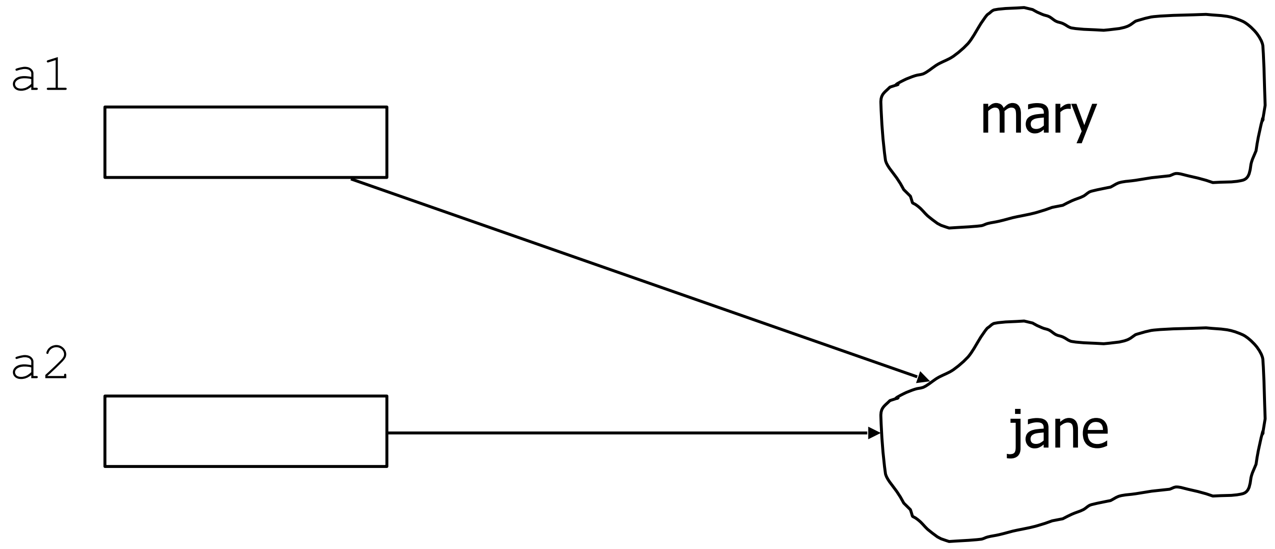
Aliasing

```
String a1 = new String("mary");
String a2 = new String("jane");

a1 = a2;
// What happens?
```



Aliasing

```
String a1 = new String("mary");
String a2 = new String("jane");

a1 = a2;
// What happens?
```



a1 and a2 are aliases

Input

- Java provides Scanner class

```
Scanner scan;
scan = new Scanner(System.in);
```

Methods

```
String next()
String nextLine()
double nextDouble()
int nextInt()
```

```
import java.util.Scanner;

Scanner s;
String reply;

s = new Scanner(System.in);
reply = s.nextLine();
System.out.println("Reply was "+reply);
```

Example Program

```
import java.util.Scanner;

public class GasMileage
{
   public static void main (String[] args)
   {
      int miles;
      double gallons, mpg;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter the number of miles: ");
      miles = scan.nextInt();

      System.out.print ("Enter the gallons of fuel used: ");
      gallons = scan.nextDouble();

      mpg = miles / gallons;

      System.out.println ("Miles Per Gallon: " + mpg);
   }
}
```

Informal HW

- Get the JDK on your computer

  Test it by running

  ```
  javac -version
  ```

  in a command shell

- Get JGrasp if you'd like to

Learning Objectives

- Java's edit-compile-run loop
- Basics of object-oriented programming
- Classes
  - objects, instantiation, methods
- Primitive types
- Math expressions
- Output, input
- Strings

Next Time

- Control flow
- Arrays
- Classes
  - Instance data
  - Methods
  - Visibility
  - Inheritance