

# COMMAND LINE ARGUMENTS

---

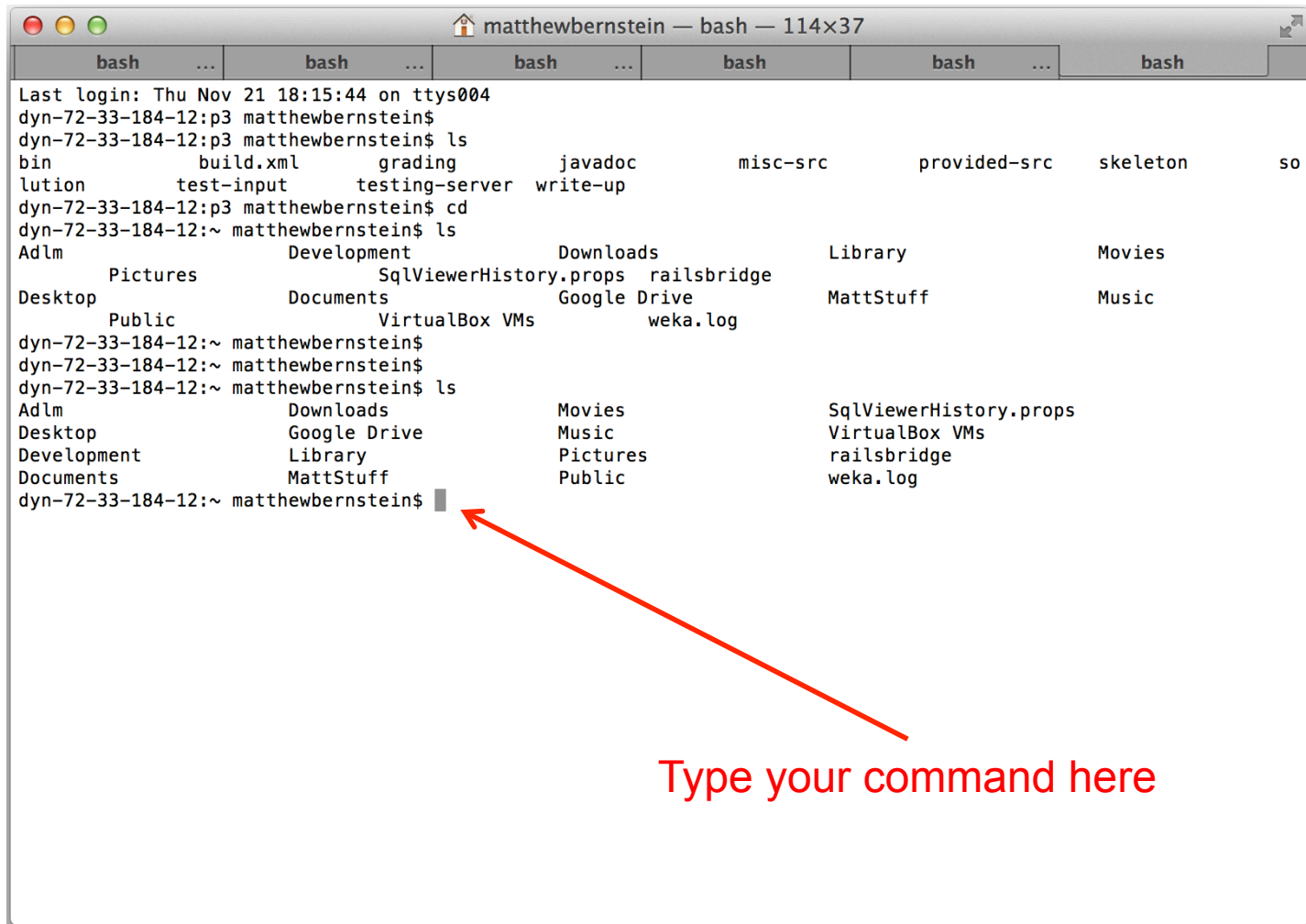
CS302 – Introduction to Programming  
University of Wisconsin – Madison  
Lecture 27

By Matthew Bernstein – [matthewb@cs.wisc.edu](mailto:matthewb@cs.wisc.edu)

# Development Environments

- There are many environments in which you can choose to write programs
- You are familiar with the environment in a program called Eclipse
- Eclipse is an Integrated Development Environment (IDE)
- It is a big program that has many tools for building and testing your programs
- If you don't have an IDE, then you will most likely use a **command shell window** such as the **terminal**
- The terminal is a program that use to interact with your computer through text

# Command Shell Window



```
matthewbernstein — bash — 114x37
bash ... bash ... bash ... bash bash ... bash
Last login: Thu Nov 21 18:15:44 on ttys004
dyn-72-33-184-12:p3 matthewbernstein$
dyn-72-33-184-12:p3 matthewbernstein$ ls
bin          build.xml    grading      javadoc      misc-src     provided-src  skeleton     so
lution     test-input  testing-server write-up
dyn-72-33-184-12:p3 matthewbernstein$ cd
dyn-72-33-184-12:~ matthewbernstein$ ls
Adlm          Pictures      Development  Downloads    Library      Movies
Desktop       Public        Documents    SqlViewerHistory.props railsbridge  MattStuff    Music
VirtualBox VMs          weka.log
dyn-72-33-184-12:~ matthewbernstein$
dyn-72-33-184-12:~ matthewbernstein$
dyn-72-33-184-12:~ matthewbernstein$ ls
Adlm          Downloads    Movies       SqlViewerHistory.props
Desktop       Google Drive Music        VirtualBox VMs
Development   Library      Pictures     railsbridge
Documents     MattStuff   Public       weka.log
dyn-72-33-184-12:~ matthewbernstein$ █
```

Type your command here

# Running Programs from the Command Line

- You can run programs from your command line
- If we have a program called “SheepMaster”, we would run it with the following command

>> SheepMaster

Command Prompt

Command

# Passing Arguments to Your Program

- You can pass arguments to your program when invoking you program from the command line:

```
>> SheepMaster -EasyMode
```



The argument is “-EasyMode”

# Specifying Files

- One of the common uses of command line arguments is the ability of the user to specify which file they would like their program to read from
- For example, let's say we have a program called "ReadFile" and it requires us to pass it a file path that refers to the file it will read from, we would do this as follows:

```
>> ReadFile /folder/myFile.txt
```

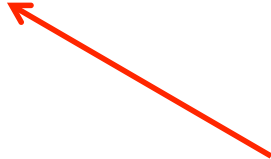


Path to the file, "myFile.txt"

# Receiving Command Line Arguments

- In Java, we receive the command line arguments in the **main** method.
- The command line arguments are passed as an array of Strings
- Remember the main method header:

```
public static void main(String[] args)
{
    ...
}
```



“args” will store all of the command line arguments

# Example

- Let's say we have a program called "CmndLine" and we invoke it as follows:

```
>> CmndLine hello world whatsup
```

- In our main method, the `String[] args` variable will store the arguments:

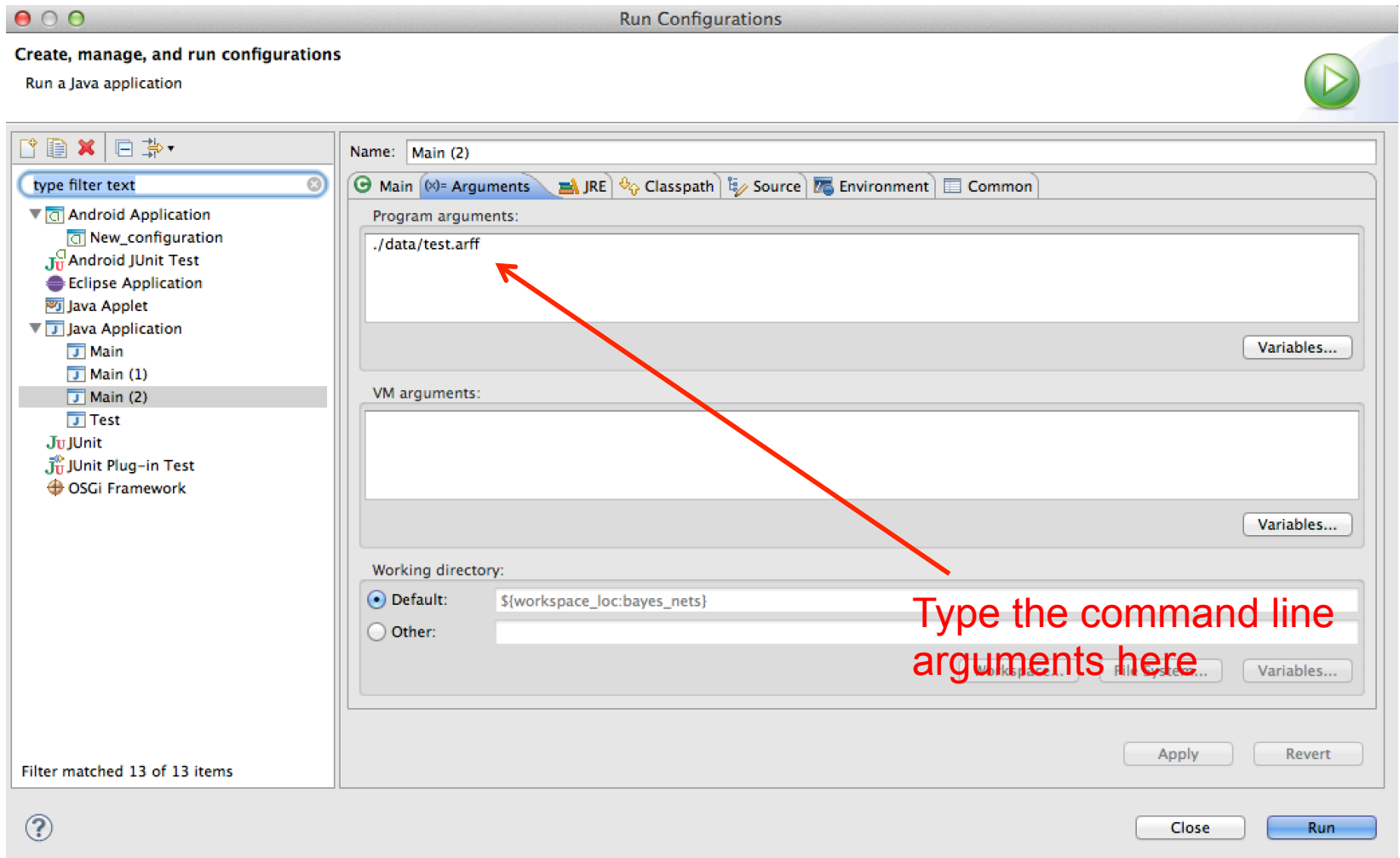
```
["hello", "world", "whatsup"]
```



# Command Line Arguments in Eclipse

- Since Eclipse does not have a command line, it has a feature that allows you to specify the command line arguments you want to pass to your program when you run it in Eclipse:
- Run → Run Configurations → Arguments
- Then type the the arguments into the textbox labeled “Program arguments:”
- Every time you run your program, Eclipse will pass the arguments specified in these configurations

# Command Line Arguments in Eclipse



The screenshot shows the Eclipse IDE's "Run Configurations" dialog. The "Name" field is set to "Main (2)". The "Program arguments" field contains the text `./data/test.arff`. A red arrow points from the text "Type the command line arguments here" to this field. The "Working directory" is set to `${workspace_loc:bayes_nets}`. The "Run" button is highlighted in blue.

Run Configurations

Create, manage, and run configurations

Run a Java application

Name: Main (2)

Program arguments: `./data/test.arff`

VM arguments:

Working directory:

Default: `${workspace_loc:bayes_nets}`

Other:

Type the command line arguments here

Apply Revert

Close Run

# Validating the Correct Command Line Arguments

- If your program requires that the user enter command line arguments according to some specification, then it is your job as the program's writer to validate that the user entered arguments of the correct format
- For example, we should always check that the user is entering the correct number of arguments
- If the user is not using your program correctly, then you should print a message that tells the user how to correctly invoke your program

# Example

- If your program requires the user include a file path and the user does not include such an argument, your program should print something like:

Usage:

ReadFile <file path>



Instructions telling the user how to invoke the program

# Example

- Your program would look something like:

```
public static void main(String[] args)
{
    if (args.length < 1)
    {
        System.out.println("Usage:");
        System.out.println("ReadFile <file path>");
        return;
    }
}
```


...

# Converting Strings to Numbers


- Let's say we need to convert a command line argument from a string to a double
- How do we do this?
- Answer: Use methods in the wrapper classes
- Example:

```
double price = Double.parseDouble( args[1] );
```

Using the static method  
**parseDouble**  
That belongs to the **Double**  
class



Passing the 2<sup>nd</sup> command  
line argument that was  
provided by the user



# Other Conversions

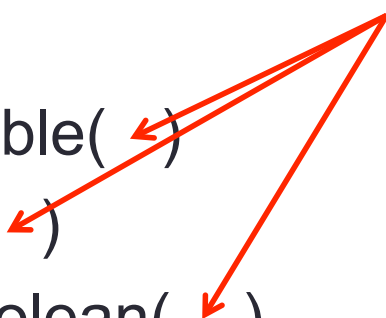
- Each wrapper class has a “parse” method that accepts a String object and returns an object that is the same type as the wrapper class you are using to invoke the method
- Examples:

Double.parseDouble( )

Integer.parseInt( )

Boolean.parseBoolean( )

All of these methods  
are passed a String  
to be parsed



# Catching an Exception

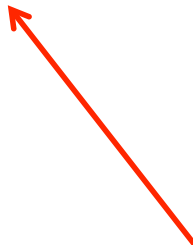
- Whenever you call one of these “parse” methods, you need to surround it with a try-catch block in order to catch the exception that is thrown in the event that the String cannot be parsed



# Catching the Exception

```
int value;  
  
try  
{  
    value = Integer.parseInt(args[0]);  
}  
catch( NumberFormatException e )  
{  
    e.printStackTrace();  
}
```

We need to catch  
a NumberFormatException



# Cool CS Link

- JMonkeyEngine - a development platform for creating video games in Java
- <http://jmonkeyengine.org/>

