Objectives

- Testing
- Collaboration
- Coverage

1

Review

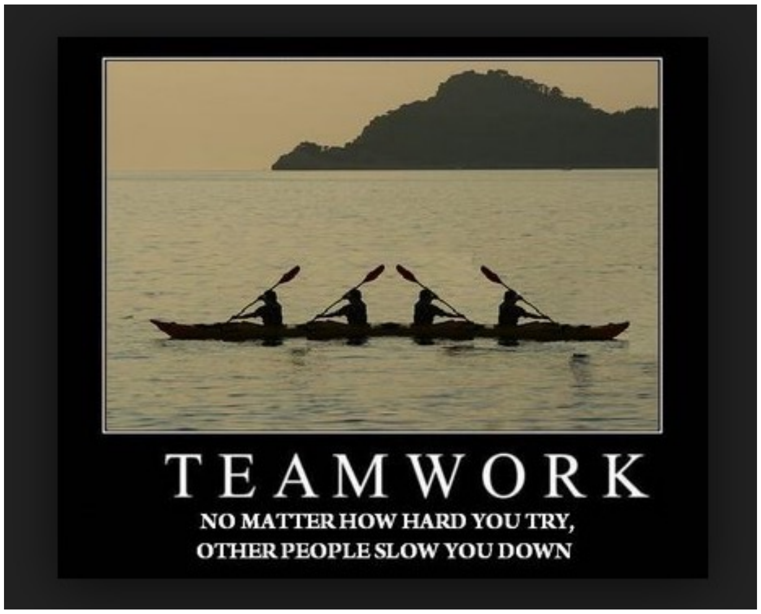
1. What are the steps to a JUnit test case?
   - How do we implement them?
2. What approaches did you take to writing good test cases to reveal the mutants?
   - How is programming *tests* the same/different from programming *generally*?
3. What are the benefits of unit testing/using JUnit?
   - Consider if you were developing/maintaining the `thirdShortest` method
   - How would your testing/development process change?
4. Is it okay that some mutants passed some of the test cases?
5. Recall the characteristics of good unit tests
   - How did you achieve them in your testing?
6. True or False. Unit testing is all the testing that needs to be done for an application.
7. Why did the output from `RevealingMutantsEvaluator` come out in strange/unexpected orders sometimes?

2

Catch the Mutants: Post-Mortem

- One test case can find multiple bugs
  - There is not a one-to-one mapping
- There was an issue in Mutants 11 and 12 that they had the same bug as Mutant 9 plus another bug ☹

3



4

Think about Team (Group) Projects

- Why did some work well?
- Why were some disasters?

Teams Work Best When They are **Interdependent**

- In code terms, we want *loose coupling*
  - Depend on each other but don't depend on their details
- Consider
  - Are you allowing your team to truly be interdependent?
  - Who might be you be ignoring?
  - Who might be allowing themselves to feel inadequate?
  - How do you show appreciation for each other and yourself?

Collaboration: Team Project

- Version Control does not eliminate need for communication
  - Process becomes much more difficult if developers do not communicate
- Keep the version to be graded in `main` branch
- Before picking up again on development, **pull** the repository
  - Get others' changes in main; merge into your branch
- Each student on team must make *significant* commits to the project's repository

Collaboration: Team Project

- Need to talk about the solution
- Discuss your plan, e.g.,
  - Your assumptions about the Car class
  - Your system for testing to make sure that you test everything
  - Organization of test cases
  - Naming
  - Division of labor
- Maintain planning documents too
  - in GitHub or elsewhere

Collaboration: Workflow – Seeking Feedback

1. Create a branch for your work from main
   - Commit periodically
   - Write descriptive comments so your team members know what you did and why
2. Push your branch
3. Open a ***Pull Request*** on your branch in GitHub
   - You can tag your teammates to let them know that you've completed your work
   - Team: discuss and review potential changes – can still update
4. Merge pull request into `main` branch (when ready)
5. Pull the `main` branch to get the latest code
   - May want to merge main into your branch

Collaboration: Workflow

1. Create a branch for your work
   - Commit periodically
   - Write descriptive comments so your team members know what you did and why
2. Switch to `main`
3. Pull `main` branch
4. Merge your branch into the `main` branch
   - Handle merge conflicts
   - Commit
5. Push `main` branch

Guidance for Writing JUnit Tests

- A test method should focus on one behavior
  - If test case fails, the test case should be helpful in narrowing down where the problem is
- Testing isn't typically "creative" and doesn't need to be generalizable
  - Code should be straightforward
- See examples linked from course schedule page

Guidance for Organizing JUnit Tests

- Group tests in methods, classes
- Classes could be distinguished by behavior, by error conditions, by set up method…
- Name methods based on what they test
  - Template: `functionality_state_expectedresult`
  - Example: `go_fulltank_moves`

Suggestions for How to Approach

- THINK first
  - Use paper/a document to structure your thoughts and systematically consider what you need to test
  - Decide on assumptions about specification
- Iterative approach
  - Each team member implement a few tests
    - Put into repository
  - Discuss as a team
    - Reconsider/confirm your assumptions, how you want to break up the work

Nov 2, 2022 Sprenkle - CSCI209 13

13

Review: Software Testing Issues

- How should you test? How often?
  - Code may change frequently
  - Code may depend on others' code
  - A lot of code to validate
- How do you know that an output is correct?
  - Complex output
  - Human judgment?
- What caused a code failure?

➥ Need a *systematic*, *automated*, *repeatable* approach

Nov 2, 2022 Sprenkle - CSCI209 14

14

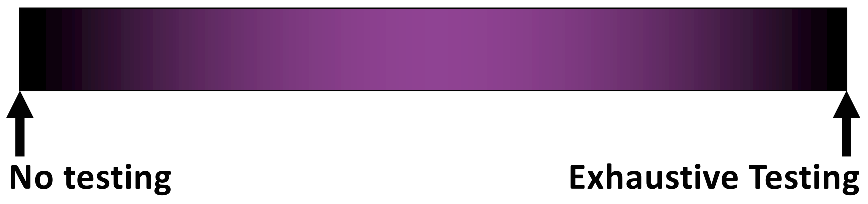
Software Testing Issues

- How do we know if our code is correct?
  - How do we know that we’ve exposed all the faults?
  - How confident are we in its correctness?
- How do we know if we’ve tested enough?
  - Passes all of our TDD test suite
    - But did we come up with *all* the necessary test cases?
  - Time? It’s been a couple hours/days/…
  - Number of test cases executed? A lot!
  - I asked my sister and she’s really smart and she says that it’s enough

Software Testing Issues

- How do we know if our code is correct?
  - How do we know that we’ve exposed all the faults?
  - How confident are we in its correctness?
- How do we know if we’ve tested enough?
  - Our customers want this product soon but we need product to be correct
    - Harder to fix after it has been released

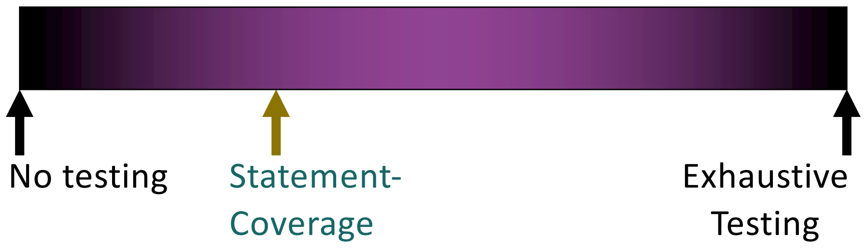
Testing Continuum



- Give to customer immediately
- Likely buggy!
- Very little confidence in program's quality

- Test *every possible* input
- Costly, impractical
- Need to release application to customers *sometime!*

Testing Continuum



- Need to execute **all code**
- Cover (i.e., execute) all **statements** in the program

Analogy: Map coverage

Goal: Expose all the "scarecrows"

19

Statement Coverage

- Cover all statements in the program

Test Suite: num=5

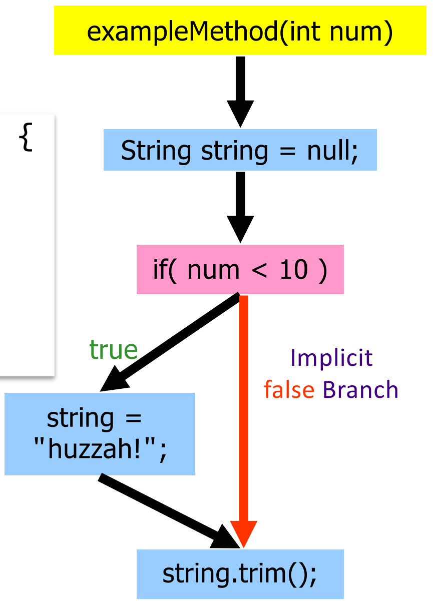
```
  public String exampleMethod(int num) {
✓ 1   String string = null;
✓ 2   if (num < 10) {
✓ 3       string = "huzzah!";
      }
      // remove leading & trailing whitespace
✓ 4   return string.trim();
  }
```

Is this method bug-free?

20

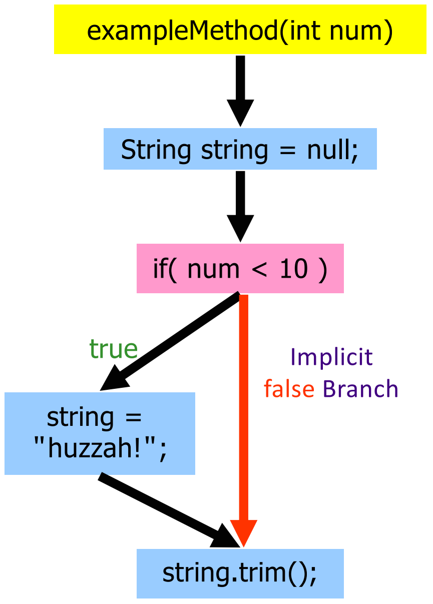
Program Flow

```
public String exampleMethod(int num) {
    String string = null;
    if (num < 10) {
        string = "huzzah!";
    }
    return string.trim();
}
```



What Went Wrong?

- Test suite had 100% statement coverage but missed a **branch/edge**
- Try covering all **edges** in program's flow
  - Also covers all **nodes**
  - Called **Branch Coverage**



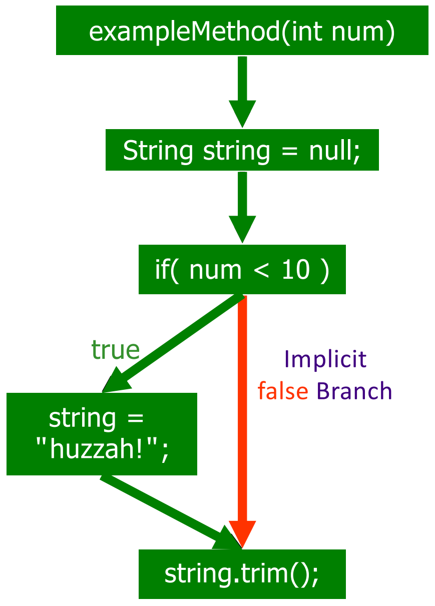
Branch Coverage

- Cover all **branches** in the program

Test Suite:

num=5,

num=10



Nov 2, 2022 Sprenkle - CSCI209 23

23

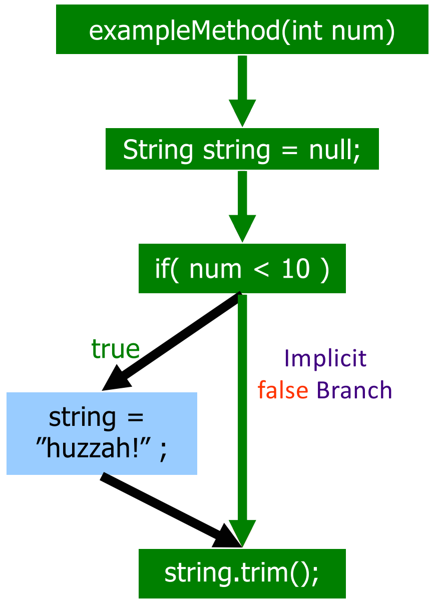
Branch Coverage

- Cover all **branches** in the program

Test Suite:

num=5,

num=10



Nov 2, 2022 Sprenkle - CSCI209 24

24

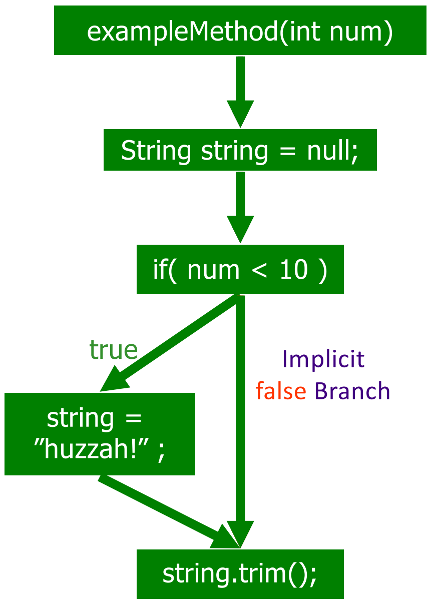
Branch Coverage

- Cover all **branches** in the program

Test Suite:

num=5,

num=10



25

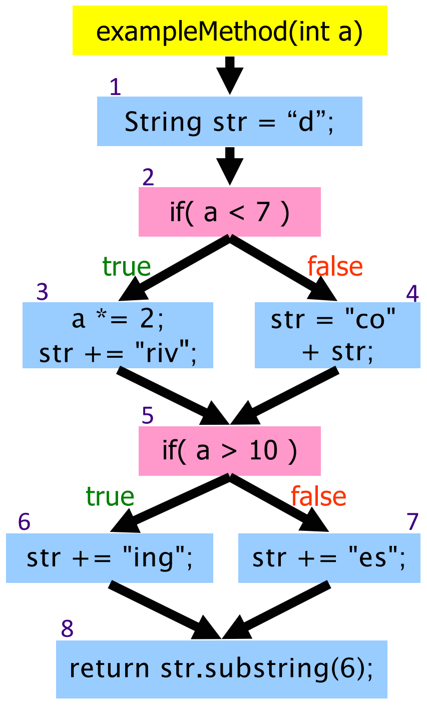
Example 2

```
public static String exampleMethod(int a) {
   String str = "d";
   if ( a < 7 ) {
      a *= 2;
      str += "riv";
    } else {
      str = "co" + str;
    }

    if( a > 10 ) {
      str += "ing";
    } else {
      str += "es";
    }
   return str.substring(6);
}
```

26

Example 2

```
public String exampleMethod(int a) {
   String str = "d";
   if ( a < 7 ) {
      a *= 2;
      str += "riv";
    } else {
      str = "co" + str;
    }

    if( a > 10 ) {
      str += "ing";
    } else {
      str += "es";
    }
   return str.substring(6);
}
```



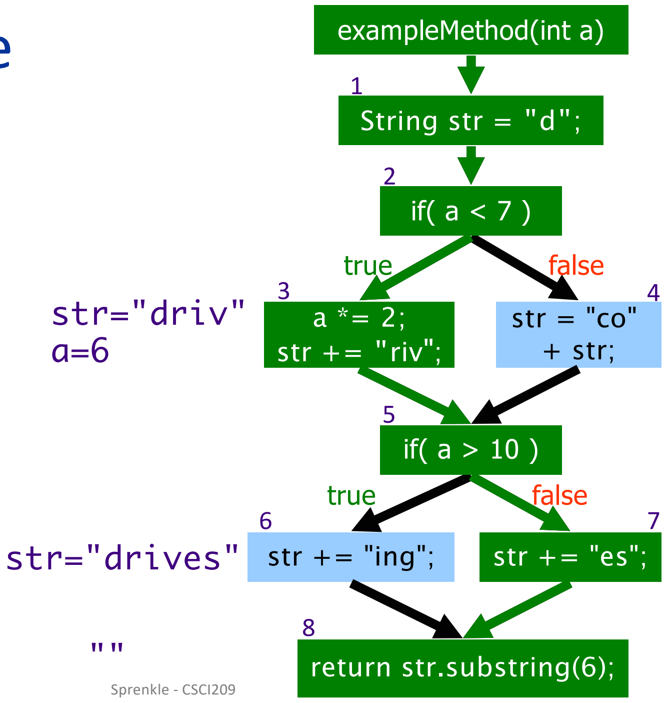
27

Branch Coverage

Test Suite:

a=3,

a=30



28

Branch Coverage

Test Suite:

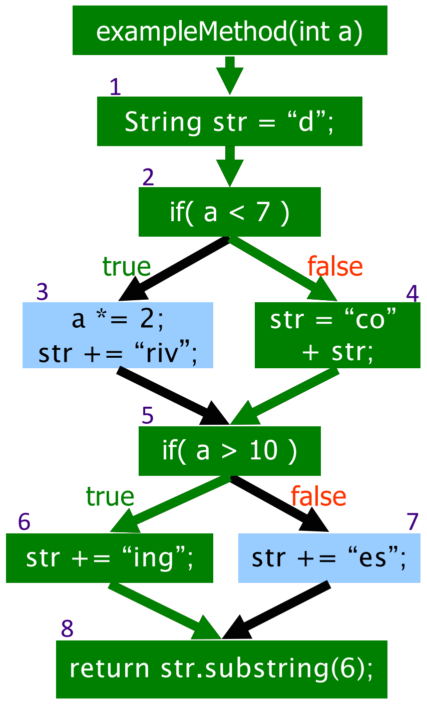
a=3,

a=30

str="cod"
a=30

str="coding"

""



29

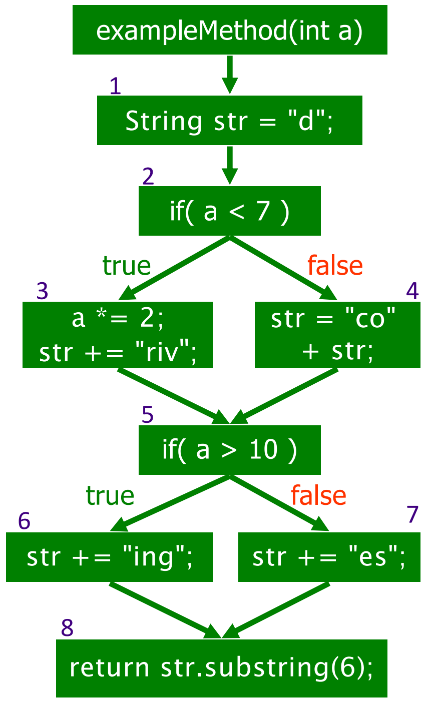
Branch Coverage

Test Suite:

a=3,

a=30

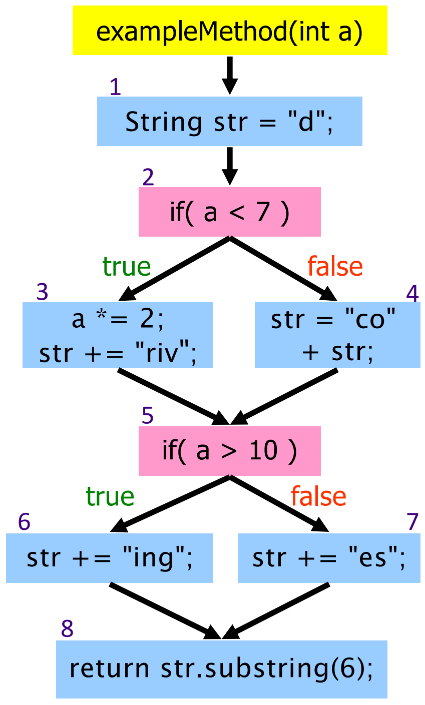
Is this method bug free?



30

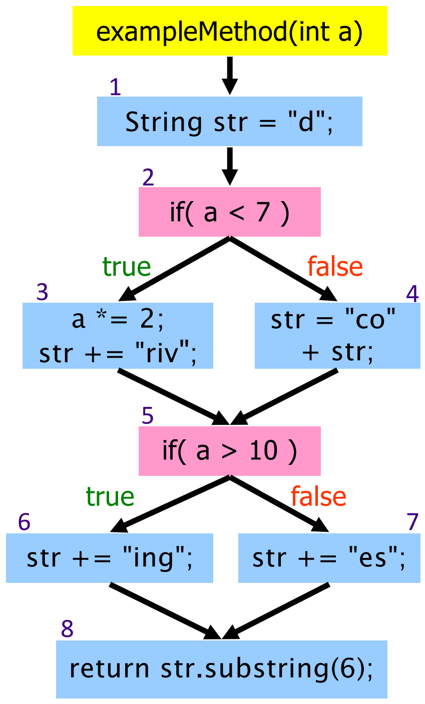
What Went Wrong?

- Test suite had 100% branch (and statement) coverage but missed a **path**
- Try to cover all **paths** in program's flow
  - Also gets all **branches**, **nodes**
  - Called **Path Coverage**



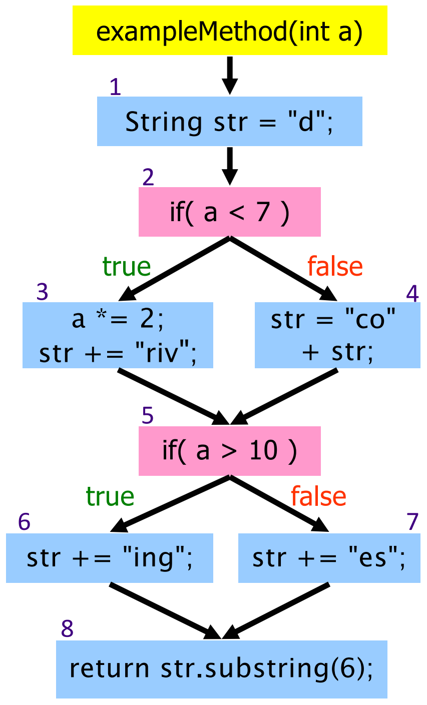
Path Coverage

- Cover all **paths** in program's flow
- How many paths through this method?
- What test cases would give us path coverage?



Path Coverage

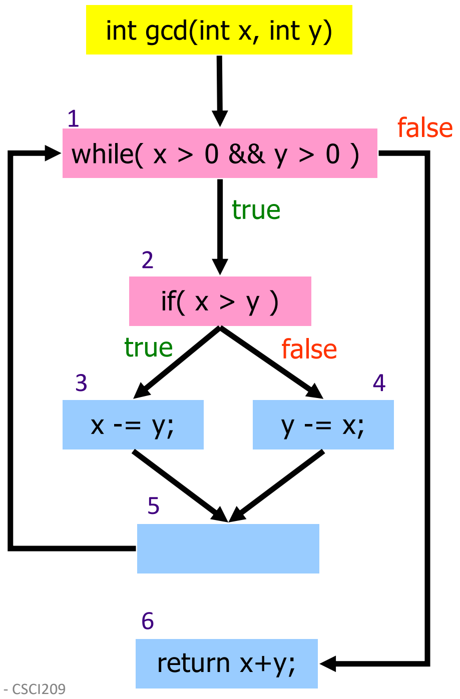
- Cover all **paths** in program's flow
- How many paths through this method? 4
  - 1-2-**3**-5-**6**-8
  - 1-2-**3**-5-**7**-8
  - 1-2-**4**-5-**6**-8
  - 1-2-**4**-5-**7**-8
- What test cases would give us path coverage?
  - One possibility: a = 3, 30, 6, 10



33

Example 3

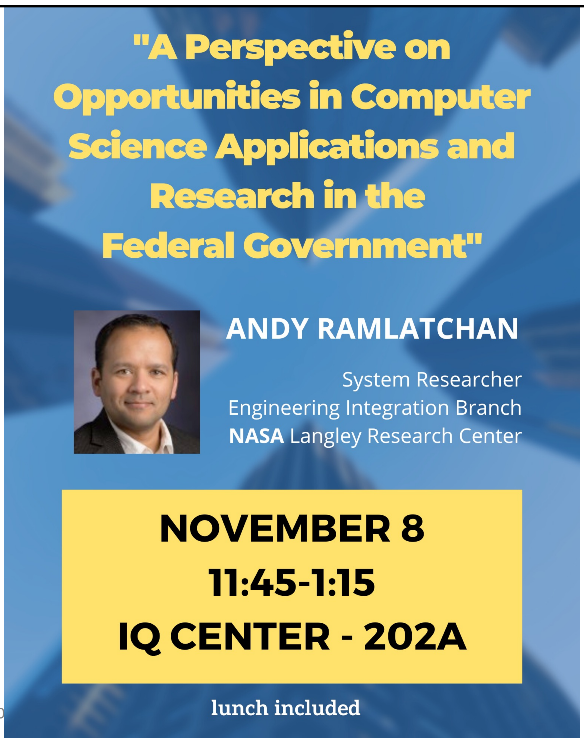
```
/**
 * Euclid's algorithm to calculate
 * greatest common divisor
 */
public int gcd( int x, int y ) {
    while ( x > 0 && y > 0 ) {
        if( x > y ) {
                  x -=y ;
        } else {
                  y -=x;
        }
    }
    return  x+y;
}
```



34

Looking Ahead

- Friday
  - ➢ Coverage, Design
- Wednesday: Testing project



"A Perspective on Opportunities in Computer Science Applications and Research in the Federal Government"

ANDY RAMLATCHAN

System Researcher
Engineering Integration Branch
NASA Langley Research Center

NOVEMBER 8
11:45-1:15
IQ CENTER - 202A

lunch included