CSE 331

Software Design & Implementation

Section: Java Tools; Integers

Reminders

- HW2 setup is important! See Panopto recordings on Canvas

Upcoming Deadlines

- HW2 due 11pm tonight (6/29)
- Prep. Quiz: HW3 due 11pm Tuesday (7/03)

Last Time…

- Specifications
- Abstract Data Types (ADTs)

Today's Agenda

- Java Tools
- Demo: Setup
- Integers and Bases

HW2 Overview

- HW2 has a few different pieces—make sure to do them all!
  - Written portion (submit on Gradescope)
    - Reasoning with loops
  - Coding portion (submit with GitLab tag)
    - Setting up repo, simple Java code
    - Looking at JUnit tests
    - Debugging code
    - Implementing code based on an invariant

The written portion can be done before you setup the software.

Course resources

- We can't cover everything in an hour

- Read documentation: cs.uw.edu/331 > "Resources" tab
  - "Project Software Setup"
  - "Editing, Compiling, Running, and Testing Java Programs"
  - "Version Control (Git) Reference"
  - "Assignment Submission"

- The resources page is a treasure trove of helpful information!

Software You Need

- Java 17
  - adoptium.net/temurin/archive
  - Choose “OpenJDK 17” and install **jdk-17.0.7+7** with the **JDK installer** for your OS
  - Windows: Select ”Add to PATH” and “Fix Registry” during install
- IntelliJ
  - jetbrains.com/idea
  - Recommended: Ultimate version
    - Comes in handy later in the course
    - Free for students, see course website for link to license
  - **Install the latest version**
- Git
  - git-scm.com
  - (Might be slightly newer version than the XCode command line tools on macOS if you have those installed)
  - Comes with Git Bash on Windows – important!

Warning: You must use JDK 17

- Must use JDK version 17
  - Be sure that's what you have installed!
  - Download links in Resources webpage
  - Use the Adoptium installers (only)

- An out-of-date JDK can lead to very confusing bugs
  - No fun for either of us!

Version control

- Also called source control, revision control
- System to track changes in a project codebase
  - Unit of change ~ lines inserted/deleted across some files
- Essential for managing software projects
  - Maintain a history of code changes
  - Revert to an older project state
  - Merge changes from multiple sources

- We'll use **git** and **GitLab** in this course, but alternatives exist
  - Subversion, Mercurial, CVS
  - Email, dropbox, thumbdrives (don't even think of doing this!)

Version control concepts

- A **repository** (“repo”) stores a project’s entire codebase
  - Stored in multiple places and synchronized over the internet
  - Tracks the files themselves and changes to them over time

- Each developer **clones** her own **working copy** of the repo
  - Makes a local copy of the codebase, on her laptop/computer
  - She modifies these files directly, with her IDE or text editor

- Each developer **commits** changes to her working copy
  - Saves “a commit” to version control history
  - Affects only the local working copy
  - Must synchronize with remote repo to share commits each way

Essential git concepts

- **`commit`**
  - Saves (a subset of) the changes to the local repository
  - Has a brief message summarizing changes

- **`push`**
  - Sends local commits to the repository (on GitLab)
  - Allows other computers to then "pull" those commits/changes, see below.

- **`pull`**
  - Synchronizes working copy to match the remote repository
  - **`clone`** = the first pull, also sets up the repository for the first time

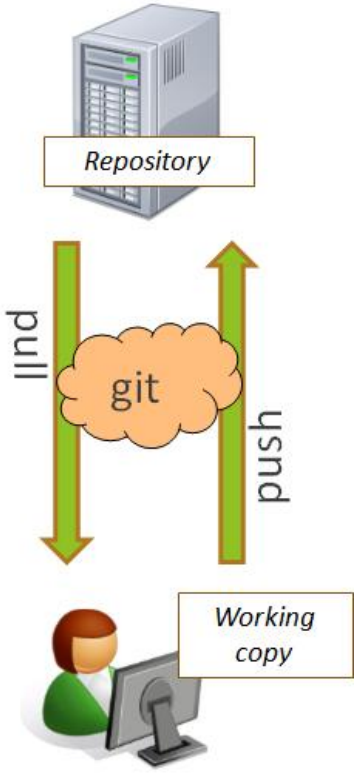
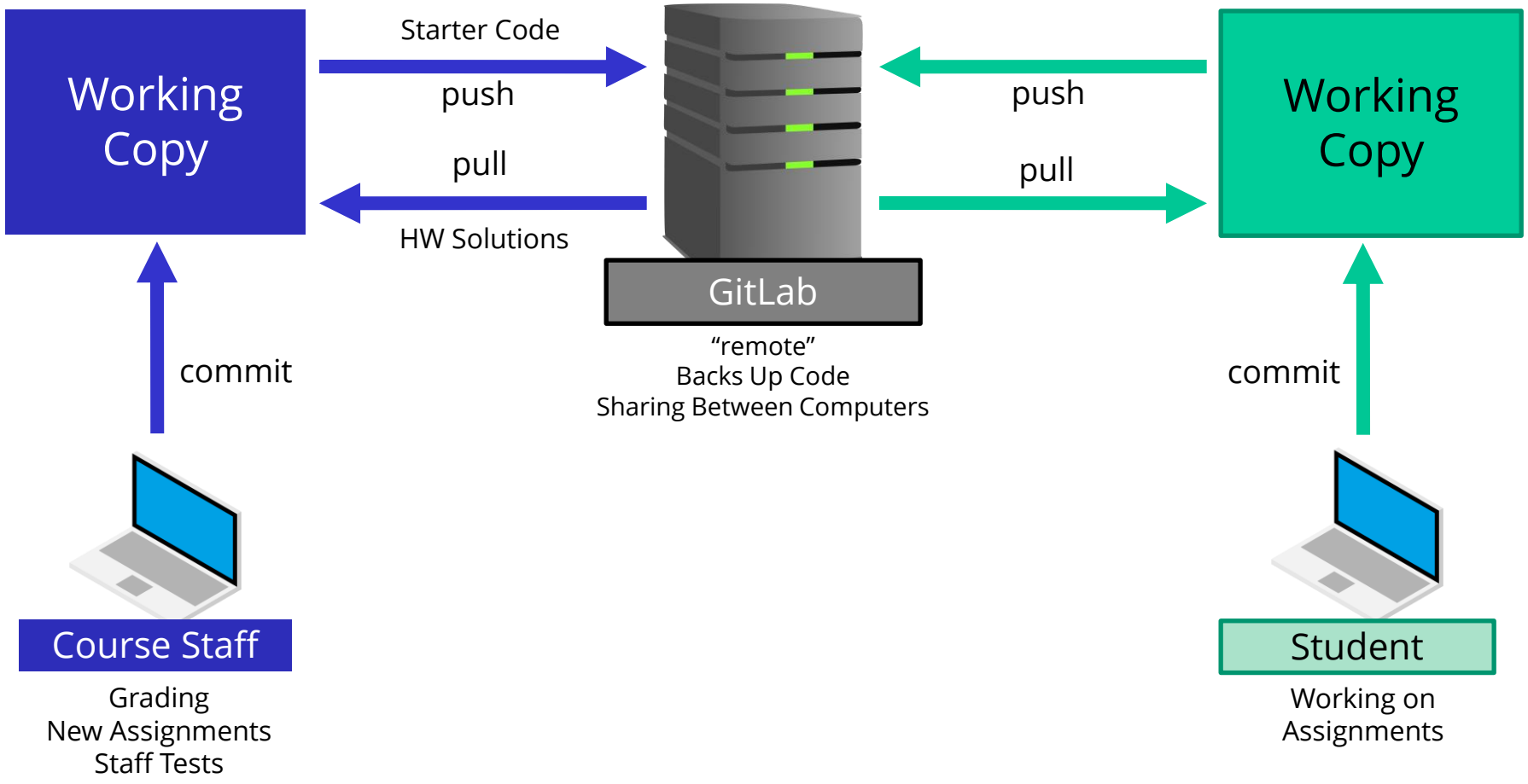


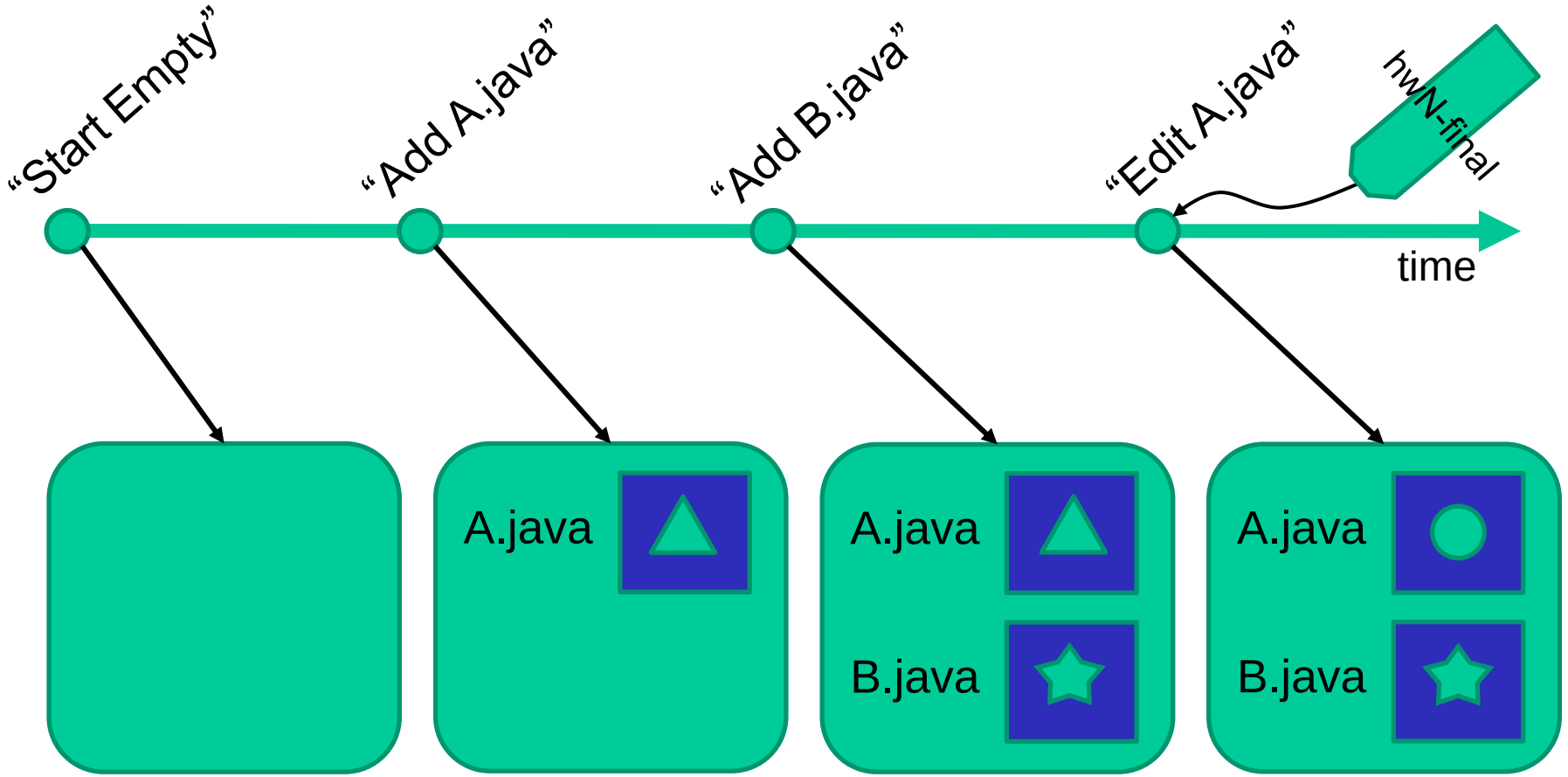
Diagram of git usage



Your GitLab repository

- We will push starter code to your repo for each homework
  - After HW2, you'll get it by pulling
- Commit and push your code as you do the assignment
  - Recommended process: edit, test, pull, commit, push
- Submit homework ***N*** by creating a tag "**hw*N*-final**"
  - *Check that you've committed and pushed all your work* ***before you tag!***
  - Do **not** attach a message with the tag
  - Example: "**hw2-final**" for HW2
- Without the right tag, your homework might not be graded!

Example commit history



Gradle: what is it

- Gradle is a tool for build automation
  - Simplifies compiling, running, and testing a software project
  - No need to install: included in the starter code!

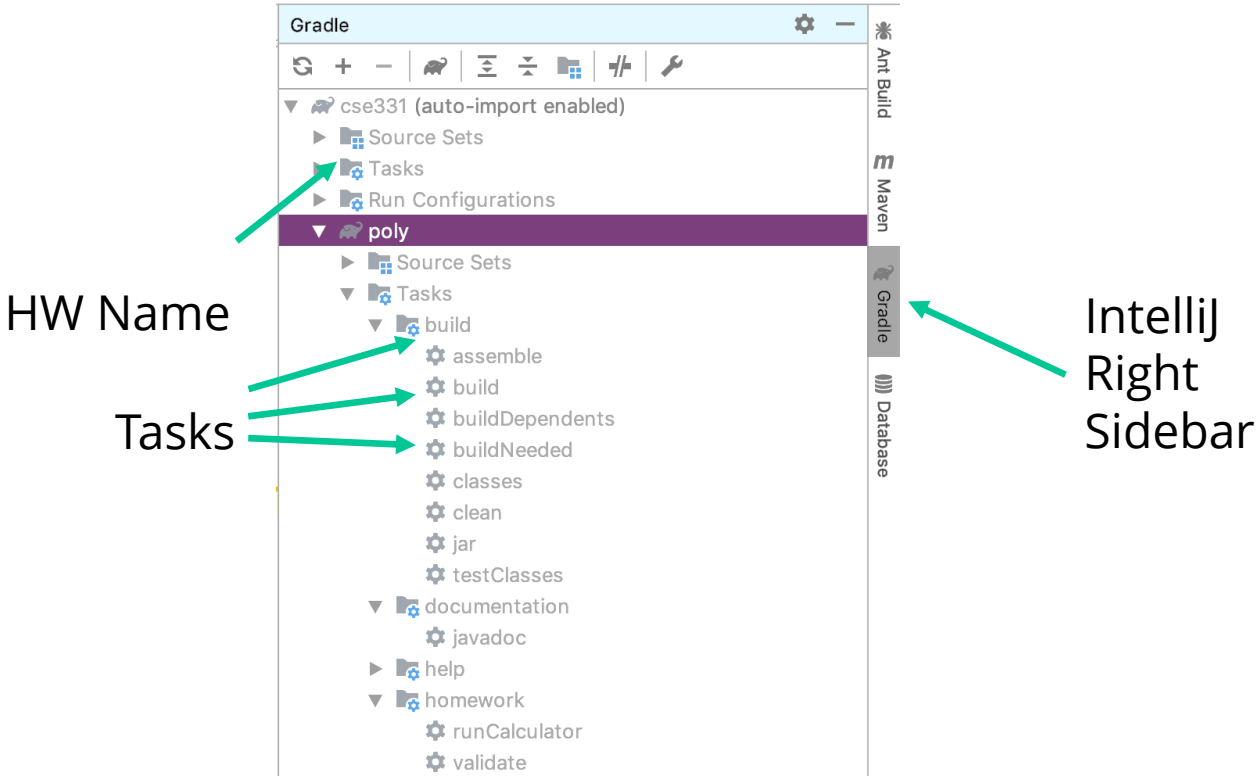
- Configured by the file **build.gradle** (and others) in your repo
  - You shouldn't modify this (can interfere with grading)!
  - Ask the course staff for help if it got messed up accidentally.

- IntelliJ has built-in support to work with Gradle

- Gradle is how you run/validate your code on attu

Gradle: how to use it

- You can use Gradle at the command line or in IntelliJ (recommended)
  - Every homework assignment has a “name” – HW3 is “hw-setup”



IntelliJ Gradle Panel

- Double-click tasks to run them.
- Make sure you’re in the **right assignment’s task** list, each one has its own tasks.

Development Workflow (Pipeline)

In general, only do this at the end of an assignment, but let's see how it works with a practice tag.

1. Create the tag with the correct name. For now, use **section-demo**. See assignment specs for the tags to use for each assignment.
   - **Git** > **New Tag**
   - Enter a tag name. (**Tags are case-sensitive.**)
   - **DO NOT** include a message. (This can make the tags difficult to move later, if you need to.)
   - Tags are automatically attached to the current commit on the remote repository (so **you need to create tags *after* creating and pushing the commit you want to tag in a *separate transaction***).
2. "push" the changes to tell GitLab about the tag (so the staff can see it!)
   - **Git** > **Push**
   - Make sure "Push Tags" (bottom left) is **checked**. (Choose "All")

Development Workflow (Pipeline)

GitLab Runners:

- Triggered when you push the tag
  - Don't see a runner? Make sure you have the right tag name! (Tags are case-sensitive)
- Runs some sanity checks (build, javadoc, and your tests) to look for common errors.
- If your runner fails, you should ***definitely*** fix it, then move the tag and check the runner again. This ensures that the staff will be able to get your code to grade
- Open your GitLab project online, go to CI/CD Pipelines (found in left hand options bar)
- For **section-demo**, you'll see a message and the pipeline should pass.
- For actual assignments, you'll see it run checks on your assignment, then it'll either pass or fail and print an error message on failure.
- Can also tag and remove tags via GitLab GUI if it is easier.
- Remember, just because the pipeline passes *does not* mean the autograder will pass. It just means that we will be able to grade your homework

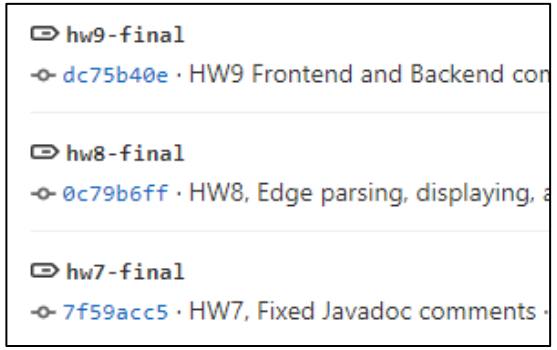
Development Workflow Demo (6)

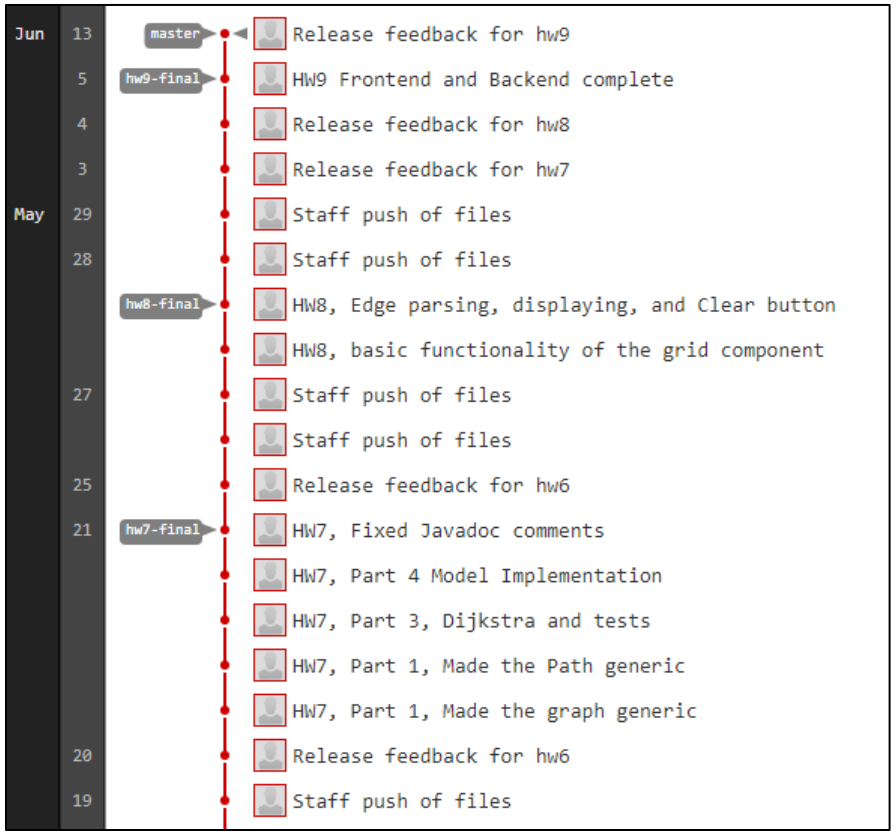
Verifying your tag is on the correct commit:

- GitLab Repository: **Left Sidebar > Repository > Graph**

This page provides a good visual for which commit your tags are attached to!

Also can check out **Repository >Tags** (browse the files and check that the SHA matches the one found in **Repository > Commits**)





HW3

In HW3, you will be writing methods in the **Natural** class

Let's look at the specification:

```
/**
 * Represents an immutable, non-negative integer value
 * along with a base in which to print its digits, which we
 * can think of as a pair (base, value).
 * For example, (2, 5) represents the integer 5 (in decimal),
 * but it will show its digits as 101 (in binary) when
 * printed.
 *
 * We require that the base is at least 2 and at most 36 for
 * simplicity.
 */
public class Natural { … }
```

(Pipeline)

Converting Between Bases

1) Take the biggest multiple of a power of the new base you can fit.

2) Divide number by base power. This tells you how many times it can fit in, which becomes your first digit (most significant digit)

3) Find the next biggest power of the new base you can fit in the remainder

Different Base Examples

Let's take the value **10.** We can use the constructor:

```
public Natural(int base,
int value){…}
```

`new Natural(10, 10)` => "10"

`new Natural(2, 10)` => "1010"

`new Natural(3, 10)` => "101"

`new Natural(4, 10)` => "22"

Convert 80 (base 10) to base 6:

- Largest power of 6 that fits: 6^2 = 36
- 36 fits 2 times (36 x 2 = 72), so first digit is 2.
- Remainder 80 – 72 = 8

- Largest power of 6 that fits: 6^1 = 6
- 6 fits 1 time (6 x 1 = 6), so second digit is 1.
- Remainder 8 – 6 = 2

- Largest power of 6 that fits: 6 ^ 0 = 1
- 1 fits 2 times (1 x 2 = 2), so third digit is 2.
- Remainder 2 – 2 = 0, we are done

= 212 (base 6)

`Natural` Fields

Now, let's look at the fields, RI, and AF:

```
// Shorthand: b = this.base, D = this.digits, and
//            n = this.digits.length
//
//   RI: 2 <= b <= 36 and D != null and n >= 1 and
//       if n > 1, then D[n-1] != 0 (no leading zeros) and
//       for i = 0 .. n-1, we have 0 <= D[i] < b
//
//   AF(this) = (b, D[0] + D[1] b + D[2] b^2 + ... +
//                  D[n-1] b^{n-1})

private final int base;
private final int[] digits;
```

Least significant digits come **first** in the array

`new Natural(2, 10)` => [0, 1, 0, 1] => "1010"

`leftShift()`

Now let's take a look at the left shift method:

```
/**
 * Produces a number whose digits, in this base, are the result of taking the
 * digits of this number and shifting them to the left m positions, writing
 * zeros in the now empty positions.
 * @return (this.base, this.value * this.base^m)
 */
public Natural leftShift(int m) { … }
```

How do we multiply something by 10 in base-10? Add a zero

How do we multiply something by 2 in binary? Add a zero

How do we multiply something by 100 (10^2) in decimal? Add two zeroes

What's the pattern? How can we do this in our code?

leftShift()

Now let's take a look at the left shift code:

```
public Natural leftShift(int m) {
  int[] digits = new int[this.digits.length + m];
  System.arraycopy(this.digits, 0, digits, m, this.digits.length);
  return new Natural(this.base, digits);
}
```

```
new Natural(10, 36) => [6,3] => leftShift(2)
=> ?
```

leftShift()

Now let's take a look at the left shift code:

```
public Natural leftShift(int m) {
  int[] digits = new int[this.digits.length + m];
  System.arraycopy(this.digits, 0, digits, m, this.digits.length);
  return new Natural(this.base, digits);
}
```

```
new Natural(10, 36) => [6,3] => leftShift(2)
=> [0,0,6,3] = (10,3600)
```

leftShift()

Now let's take a look at the left shift code:

```
public Natural leftShift(int m) {
  int[] digits = new int[this.digits.length + m];
  System.arraycopy(this.digits, 0, digits, m, this.digits.length);
  return new Natural(this.base, digits);
}
```

```
new Natural(10, 36) => [6,3] => leftShift(2)
=> [0,0,6,3] = (10,3600)
```

```
new Natural(2, 10) => [0,1,0,1] => leftShift(3)
=> ?
```

leftShift()

Now let's take a look at the left shift code:

```
public Natural leftShift(int m) {
  int[] digits = new int[this.digits.length + m];
  System.arraycopy(this.digits, 0, digits, m, this.digits.length);
  return new Natural(this.base, digits);
}
```

```
new Natural(10, 36) => [6,3] => leftShift(2)
=> [0,0,6,3] = (10,3600)
```

```
new Natural(2, 10) => [0,1,0,1] => leftShift(3)
=> [0,0,0,0,1,0,1] = (2,80)
```

Does this make sense?

Before next lecture…

1. Do HW2 tonight! (reminder: deadline is 11pm)
   - Written portion (submit PDF on Gradescope)
   - Coding portion (push and tag on GitLab)

2. Read documentation: cs.uw.edu/331 > "Resources" tab
   - "Project Software Setup"
   - "Editing, Compiling, Running, and Testing Java Programs"
   - "Version Control (Git) Reference"
   - "Assignment Submission"

3. Read getValue() proof in slides (recommended)

getValue()

```
public int getValue() {

  int i = this.digits.length - 1;
  int j = 0;
  int val = this.digits[i];

  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    j = j + 1;
    i = i - 1;
    val = val * this.base + this.digits[i];
  }

  // Post: val = D[0] + D[1] b + D[2] b^2 + ... + D[n-1] b^{n-1}
  return val;
}
```

What is this method doing?

Proving `getValue()`

Let's first prove that the invariant is established before the loop:

```
public int getValue() {
  {{ RI, which includes n >= 1 }}
  int i = this.digits.length - 1;
  {{ ? }}
  int j = 0;

  int val = this.digits[i];



  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  …
}
```

Proving `getValue()`

Let's first prove that the invariant is established before the loop:

```
public int getValue() {
  {{ RI, which includes n >= 1 }}
  int i = this.digits.length - 1;
  {{ n >= 1 and i = n - 1 }}
  int j = 0;
  {{ ? }}
  int val = this.digits[i];



  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  …
}
```

Proving `getValue()`

Let's first prove that the invariant is established before the loop:

```
public int getValue() {
  {{ RI, which includes n >= 1 }}
  int i = this.digits.length - 1;
  {{ n >= 1 and i = n - 1 }}
  int j = 0;
  {{ n >= 1 and i = n - 1 and j = 0}}
  int val = this.digits[i];
  {{ ? }}


  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  …
}
```

Proving `getValue()`

Let's first prove that the invariant is established before the loop:

```
public int getValue() {
  {{ RI, which includes n >= 1 }}
  int i = this.digits.length - 1;
  {{ n >= 1 and i = n - 1 }}
  int j = 0;
  {{ n >= 1 and i = n - 1 and j = 0}}
  int val = this.digits[i];
  {{ n >= 1 and i = n - 1 and j = 0 and val = D[i]}}
```

Does this imply the invariant?

```
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  …
}
```

Proving `getValue()`

Let's prove the part after the loop:

```
public int getValue() {
  …
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    …
  }
  {{ ? }}



  // Post: val = D[0] + D[1] b + D[2] b^2 + ... + D[n-1] b^{n-1}
  return val;
}
```

Proving `getValue()`

Let's prove the part after the loop:

```
public int getValue() {
  …
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    …
  }
  {{ val = D[i] b^0 + D[i+1] b^1 + … + D[n-1] b^j and i+j = n-1
     and j = n-1 }}
  ⇔ {{ ? }}


  // Post: val = D[0] + D[1] b + D[2] b^2 + ... + D[n-1] b^{n-1}
  return val;
}
```

Proving `getValue()`

Let's prove the part after the loop:

```
public int getValue() {
  …
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    …
  }
  {{ val = D[i] b^0 + D[i+1] b^1 + … + D[n-1] b^j and i+j = n-1
     and j = n-1 }}
  ⇔{{ val = D[0] b^0 + D[1] b^1 + … + D[n-1] b^{n-1} and i=0
      and j = n-1 }}

  // Post: val = D[0] + D[1] b + D[2] b^2 + ... + D[n-1] b^{n-1}
  return val;
}
```

Proving `getValue()`

Now let's prove the loop body:

```
public int getValue() {
  …
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    {{ ? }}

    j = j + 1;


    i = i - 1;


    val = val * this.base + this.digits[i];



  }
  …
}
```

Proving `getValue()`

```
public int getValue() {
  …
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    {{ val = D[i]b^0 + D[i+1]b^1 + ... + D[n-1]b^j and i+j = n-1
                                                  and j != n-1 }}
    j = j + 1;
    {{ ? }}

    i = i - 1;


    val = val * this.base + this.digits[i];



  }
  …
}
```

Proving `getValue()`

```
public int getValue() {
  …
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    {{ val = D[i]b^0 + D[i+1]b^1 + ... + D[n-1]b^j and i+j = n-1
                                                  and j != n-1 }}
    j = j + 1;
    {{ val = D[i]b^0 + D[i+1]b^1 + ... + D[n-1]b^{j-1} and i+j-1 = n-1
                                                    and j != n }}
    i = i - 1;
    {{ ? }}

    val = val * this.base + this.digits[i];



  }
  …
}
```

Proving `getValue()`

```
public int getValue() {
  …
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    {{ val = D[i]b^0 + D[i+1]b^1 + ... + D[n-1]b^j and i+j = n-1
                                                  and j != n-1 }}
    j = j + 1;
    {{ val = D[i]b^0 + D[i+1]b^1 + ... + D[n-1]b^{j-1} and i+j-1 = n-1
                                                    and j != n }}
    i = i - 1;
    {{ val = D[i+1]b^0 + D[i+2]b^1 + ... + D[n-1]b^{j-1} and i+j = n-1
                                                    and j != n }}
    val = val * this.base + this.digits[i];
    {{ ? }}


  }
  …
}
```

Proving `getValue()`

```
public int getValue() {
  …
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    {{ val = D[i]b^0 + D[i+1]b^1 + ... + D[n-1]b^j and i+j = n-1
                                                  and j != n-1 }}
    j = j + 1;
    {{ val = D[i]b^0 + D[i+1]b^1 + ... + D[n-1]b^{j-1} and i+j-1 = n-1
                                                    and j != n }}
    i = i - 1;
    {{ val = D[i+1]b^0 + D[i+2]b^1 + ... + D[n-1]b^{j-1} and i+j = n-1
                                                    and j != n }}
    val = val * this.base + this.digits[i];
    {{ (val - D[i])/b = D[i+1]b^0 + D[i+2]b^1 + ... + D[n-1]b^{j-1}
                                          and i+j = n-1 and j != n }}
    ⇔ {{ ? }}
  }
  …
}
```

Proving `getValue()`

```
public int getValue() {
  …
  // Inv: val = D[i] b^0 + D[i+1] b^1 + ... + D[n-1] b^j and
  //      i + j = n - 1, where D = this.digits,
  //      n = this.digits.length, and b = this.base
  while (j != this.digits.length - 1) {
    {{ val = D[i]b^0 + D[i+1]b^1 + ... + D[n-1]b^j and i+j = n-1
                                                  and j != n-1 }}
    j = j + 1;
    {{ val = D[i]b^0 + D[i+1]b^1 + ... + D[n-1]b^{j-1} and i+j-1 = n-1
                                                    and j != n }}
    i = i - 1;
    {{ val = D[i+1]b^0 + D[i+2]b^1 + ... + D[n-1]b^{j-1} and i+j = n-1
                                                    and j != n }}
    val = val * this.base + this.digits[i];
    {{ (val - D[i])/b = D[i+1]b^0 + D[i+2]b^1 + ... + D[n-1]b^{j-1}
                                          and i+j = n-1 and j != n }}
    ⇔ {{ val = D[i] + D[i+1]b^1 + … + D[n-1]b^j and i+j = n-1 and j != n }}
  }
  …
}
```

It's correct!