CSE 331
Software Design & Implementation

Spring 2022
Section 6 – HW6 and Midterm Review
Administrivia

• Done with HW5!

• HW6 (ADT implementation) due next week (Thurs. 5/12)

• Midterm tomorrow during lecture!

• Any questions?
Agenda

- Walk-through of the test-script driver (to run .test files)
- Managing an expensive checkRep
- Midterm review
Refresher: Format of script tests

Each script test is expressed as text-based script `foo.test`

- One command per line, of the form: `Command arg_1 arg_2 ...`
- Script’s output compared against `foo.expected`
- Precise details specified in the homework
- Match format *exactly*, including whitespace!

<table>
<thead>
<tr>
<th>Command (in <code>foo.test</code>)</th>
<th>Output (in <code>foo.expected</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CreateGraph name</code></td>
<td>created graph name</td>
</tr>
<tr>
<td><code>AddNode graph label</code></td>
<td>added node label to graph</td>
</tr>
<tr>
<td><code>AddEdge graph parent child label</code></td>
<td>added edge label from parent to child in graph</td>
</tr>
<tr>
<td><code>ListNodes graph</code></td>
<td>graph contains: label_node ...</td>
</tr>
<tr>
<td><code>ListChildren graph parent</code></td>
<td>the children of parent in graph are: child (label_edge) ...</td>
</tr>
<tr>
<td><code># This is comment text ...</code></td>
<td><code># This is comment text ...</code></td>
</tr>
</tbody>
</table>
Refresher: `example.test`

```python
# Create a graph
CreateGraph graph1

# Add a pair of nodes
AddNode graph1 n1
AddNode graph1 n2

# Add an edge
AddEdge graph1 n1 n2 e1

# Print all nodes in the graph
ListNodes graph1

# Print all child nodes of n1 with outgoing edge
ListChildren graph1 n1
```
Refresher: `example.expected`

# Create a graph
created graph graph1

# Add a pair of nodes
added node n1 to graph1
added node n2 to graph1

# Add an edge
added edge e1 from n1 to n2 in graph1

# Print all nodes in the graph
graph1 contains: n1 n2

# Print all child nodes of n1 with outgoing edge
the children of n1 in graph1 are: n2(e1)
How the script tests work

- In HW5, you wrote script tests in the form of `.test` files
  - As well as an `.expected` file for each test’s expected outcome

- The JUnit class `ScriptFileTests` runs all these tests
  - Looks for all the `.test` files in the `src/test/resources/testScripts` folder
  - Compares test output against corresponding `.expected` file

- `ScriptFileTests` needs a bridge to your graph implementation
  - That’s exactly what the `GraphTestDriver` class is for
Graph Test Driver

- `GraphTestDriver` knows how to read these test scripts

- `GraphTestDriver` calls a method to “do” each verb
  - `CreateGraph`, `AddNode`, `AddEdge` …
  - One method stub per script command for you to fill with calls to your graph code

- Note: Completed test driver should sort lists before printing for `ListNodes` and `ListChildren`
  - Just to ensure predictable, deterministic output
  - Your graph implementation itself should not worry about sorting
Graph Test Driver Output

- The Graph Test Driver is a client of our graph...
  - ...but not the only client.
  - Your graph should not be designed to be exclusively used for the test driver.

- ListChildren in the test driver should print out: “the children of parent in graph are: child(label edge) …”

- This does not mean that you should have a method on your graph called ListChildren that returns this String
  - Because that isn’t useful for other clients
Sorting with the driver

• **Use the test driver appropriately!**
  – From before: “Completed test driver should sort lists before printing.”

• Script test output for hw5 needs to be sorted so we can mechanically check it.

• This means sorted output for tests does **NOT** mean sorted internal storage in graph.
  – If sorting behavior is needed, Graph ADT clients (including the test driver) can sort those labels.
In other words…

The Graph ADT in general should **NOT** assume that node or edge labels are sorted or even comparable(!).

(of course they can be tested for equality with equals() )
Demo

Here’s a quick tour of the GraphTestDriver!
Expensive `checkReps`

- A complicated rep. invariant can be expensive to check
  - Especially iterating over internal collection(s)
  - For example, examining every edge in a graph

- A slow `checkRep` could cause our grading scripts to time-out
  - Can be really useful during testing/debugging, but
  - Need to disable the really slow checks before submitting

- We have a tension between two goals:
  - Thorough, possibly slow checking for development
  - Essential, necessarily fast checking for production/grading

- What to do?
Use a debug flag to tune `checkRep`

- Repeatedly (un)commenting sections of code is a poor solution

- Instead, use a class-level constant as a toggle
  - Ex.: `private static final boolean DEBUG = ...;
    
    - `false` for only the fast, essential checks
    - `true` for all the slow, thorough checks
  
- Real-world code often has several such “debug levels”

```java
private void checkRep() {
    assert fast_checks();
    if (DEBUG)
        assert slow_checks();
}
```
Midterm Review
intToString()

Fill in the implementation of a method that converts a **positive integer** to its **string representation in decimal** (invariant given on next slide).

{{ P: x > 0 }}

String intToString(int x)

Useful facts to recall:
1. Convert char ch that is one of ‘0’, ‘1’,…, ‘9’ to a corresponding int by doing ch – ‘0’
2. Convert int x that is one of 0,1,…,9 to a corresponding char by doing (char) (x + ‘0’)
intToString()

{{ P: x > 0 }}

String intToString(int x) {
    StringBuilder buf =
    int k = 0, y = 0;
    {{ Inv: P and buf stores the lowest k digits of x
      in reverse order and y = x / 10^k }}
    while (y != 0) {

        k = k + 1;
    }

    return buf.reverse().toString();
}
intToString()

{{ P: x > 0 }}

String intToString(int x) {
    StringBuilder buf = new StringBuilder();
    int k = 0, y = x;
    {{ Inv: P and buf stores the lowest k digits of x in reverse order and y = x / 10^k }}
    while (y != 0) {
        k = k + 1;
    }
    return buf.reverse().toString();
}
intToString()}

{{ P: x > 0 }}

String intToString(int x) {
    StringBuilder buf = new StringBuilder();
    int k = 0, y = x;
    {{ Inv: P and buf stores the lowest k digits of x in reverse order and y = x / 10^k }}
    while (y != 0) {
        k = k + 1;
    }

    return buf.reverse().toString();
}
**intToString()**

```java
{{ P: x > 0 }}
String intToString(int x) {
    StringBuilder buf = new StringBuilder();
    int k = 0, y = x;
    {{ Inv: P and buf stores the lowest k digits of x in reverse order and y = x / 10^k }}
    while (y != 0) {
        y = y / 10;
        k = k + 1;
    }
    return buf.reverse().toString();
}
```

**Inv changes k to k+1, so**
- y becomes \( x / 10^{k+1} \)
- \( y_{\text{post}} = x / 10^{k+1} = y_{\text{pre}} / 10 \)
intToString()

{{ P: x > 0 }}

String intToString(int x) {
    StringBuilder buf = new StringBuilder();
    int k = 0, y = x;
    {{ Inv: P and buf stores the lowest k digits of x in reverse order and y = x / 10^k }}
    while (y != 0) {
        y = y / 10;
        k = k + 1;
    }

    return buf.reverse().toString();
}
intToString()

{{ P: x > 0 }}
String intToString(int x) {
    StringBuilder buf = new StringBuilder();
    int k = 0, y = x;
    {{ Inv: P and buf stores the lowest k digits of x
        in reverse order and y = x / 10^k }}
    while (y != 0) {
        y = y / 10;
        k = k + 1;
    }
    return buf.reverse().toString();
}
intToString()

```java
{{ P: x > 0 }}
String intToString(int x) {
    StringBuilder buf = new StringBuilder();
    int k = 0, y = x;
    {{ Inv: P and buf stores the lowest k digits of x in reverse order and y = x / 10^k }}
    while (y != 0) {
        char ch = ?
        buf.append(ch);
        y = y / 10;
        k = k + 1;
    }

    return buf.reverse().toString();
}
```

How can we get the (k+1)-st lowest digit of x? And make it a char?
### intToString()

```java
{{ P: x > 0 }}

String intToString(int x) {
    StringBuilder buf = new StringBuilder();
    int k = 0, y = x;
    {{ Inv: P and buf stores the lowest k digits of x in reverse order and y = x / 10^k }}
    while (y != 0) {
        char ch = (char) (y % 10 + '0');
        buf.append(ch);
        y = y / 10;
        k = k + 1;
    }

    return buf.reverse().toString();
}
```
intToString() solution

{{ P: x > 0 }}

String intToString(int x) {
    StringBuilder buf = new StringBuilder();
    int k = 0, y = x;
    {{ Inv: P and buf stores the lowest k digits of x in reverse order and y = x / 10^k }}
    while (y != 0) {
        char ch = (char) (y % 10 + '0');
        buf.append(ch);
        y = y / 10;
        k = k + 1;
    }
    {{ buf stores the digits of x in reverse order }}
    return buf.reverse().toString();
}
intToString() solution

{{ P: x > 0 }}

String intToString(int x) {
    StringBuilder buf = new StringBuilder();
    int k = 0, y = x;
    {{ Inv: P and buf stores the lowest k digits of x in reverse order and y = x / 10^k }}
    while (y != 0) {
        char ch = (char) (y % 10 + '0');
        buf.append(ch);
        y = y / 10;
        k = k + 1;
    }
    {{ buf stores the digits of x in reverse order }}
    return buf.reverse().toString();
}

Why does this hold?

y = 0 => x < 10^k so x only has k digits
**Specifications**

Suppose we have a `BankAccount` class with instance variable balance. Consider the following specifications (ignore `@param`):

A. `@effects` decreases balance by amount
B. `@requires` amount $\geq 0$ and amount $\leq$ balance
   `@effects` decreases balance by amount
C. `@throws` `InsufficientFundsException` if balance < amount
   `@effects` decreases balance by amount

Which specifications does this implementation meet?

```java
void withdraw(int amount) {
    balance -= amount;
}
```
Suppose we have a `BankAccount` class with instance variable balance. Consider the following specifications (ignore `@param`):

A. `@effects` decreases balance by amount
B. `@requires` amount >= 0 and amount <= balance
   `@effects` decreases balance by amount
C. `@throws` `InsufficientFundsException` if balance < amount
   `@effects` decreases balance by amount

Which specifications does this implementation meet?

```java
void withdraw(int amount) {
    if (balance >= amount) balance -= amount;
}
```
Specifications

Suppose we have a `BankAccount` class with instance variable balance. Consider the following specifications (ignore @param):

A. @effects decreases balance by amount  
B. @requires amount >= 0 and amount <= balance  
   @effects decreases balance by amount  
C. @throws InsufficientFundsException if balance < amount  
   @effects decreases balance by amount

Which specifications does this implementation meet?

```java
void withdraw(int amount) {
    if (amount < 0) throw new IllegalArgumentException();
    balance -= amount;
}
```
Specifications

Suppose we have a `BankAccount` class with instance variable balance. Consider the following specifications (ignore `@param`):

A. `@effects` decreases balance by amount ✗
B. `@requires` amount \( \geq 0 \) and amount \( \leq \) balance ✔
   `@effects` decreases balance by amount ✔
C. `@throws` `InsufficientFundsException` if balance \( < \) amount ✔
   `@effects` decreases balance by amount ✔

Which specifications does this implementation meet?

```java
void withdraw(int amount) throws InsufficientFundsException {
    if (balance < amount) throw new InsufficientFundsException();
    balance -= amount;
}
```
Testing

Consider the `BankAccount` class again. What are some good test cases?

```java
public class BankAccount {
    /** @return current balance of account */
    public void balance() { ... }

    /**
     * @param amount to withdraw
     * @requires amount >= 0
     * @throws InsufficientFundsException
     *         if balance < amount
     * @effects decreases balance by amount
     */
    public void withdraw(int amount) { ... }
}
```

Specification test heuristic:
- amount <= balance
- amount > balance

Boundary test heuristic:
- amount = balance
- amount > balance

Others?

Should we test amount < 0?
More Reasoning

Let’s check that this method is correct.

```java
/** Return the value of this IntPoly at point x */
public int valueAt(int x) {
    int val = a[0];
    int xk = 1;
    int k = 0;
    int n = a.length - 1;
    {{ inv: xk = x^k && val = a[0] + a[1]*x + ... + a[k]*x^k }}
    while (k != n) {
        {{ ? }}
        xk = xk * x;
        {{ ? }}
        val = val + a[k+1]*xk;
        {{ ? }}
        k = k + 1;
        {{ ? }}
    }
    {{ val = a[0] + a[1]*x + ... + a[n]*x^n }}
    return val;
}
```
More Reasoning

Let’s check that this method is correct.

/** Return the value of this IntPoly at point x */
public int valueAt(int x) {
    int val = a[0];
    int xk = 1;
    int k = 0;
    int n = a.length - 1;
    {{ inv: xk = x^k && val = a[0] + a[1]*x + … + a[k]*x^k }}
    while (k != n) {
        {{ ? }}
        xk = xk * x;
        {{ ? }}
        val = val + a[k+1]*xk;
        {{ ? }}
    }
    {{ val = a[0] + a[1]*x + … + a[n]*x^n }}
    return val;
}
Let’s check that this method is correct.

```java
/** Return the value of this IntPoly at point x */
public int valueAt(int x) {
    int val = a[0];
    int xk = 1;
    int k = 0;
    int n = a.length - 1;
    {{ inv: xk = x^k && val = a[0] + a[1]*x + ... + a[k]*x^k }}
    while (k != n) {
        {{ ? }}
        xk = xk * x;
        {{ ? }}
        val = val + a[k+1]*xk;
        {{ ? }}
    }
    {{ val = a[0] + a[1]*x + ... + a[n]*x^n }}
    return val;
}
```
More Reasoning

Let’s check that this method is correct.

```java
/** Return the value of this IntPoly at point x */
public int valueAt(int x) {
    int val = a[0];
    int xk = 1;
    int k = 0;
    int n = a.length - 1;
    {{ inv: xk = x^k && val = a[0] + a[1]*x + … + a[k]*x^k }}
    while (k != n) {
        {{ inv && k != n }}
        xk = xk * x;
        {{ ? }}
        val = val + a[k+1]*xk;
        {{ ? }}
        k = k + 1;
        {{ ? }}
    }
    {{ val = a[0] + a[1]*x + … + a[n]*x^n }}
    return val;
}
```
More Reasoning

Let’s check that this method is correct.

```java
/** Return the value of this IntPoly at point x */
public int valueAt(int x) {
    int val = a[0];
    int xk = 1;
    int k = 0;
    int n = a.length - 1;
    {{ inv: xk = x^k && val = a[0] + a[1]*x + … + a[k]*x^k }}
    while (k != n) {
        {{ inv && k != n }}
        xk = xk * x;
        {{ xk = x^(k+1) && val = a[0] + a[1]*x + … + a[k]*x^k && k != n }}
        val = val + a[k+1]*xk;
        {{ ? }}
        k = k + 1;
        {{ ? }}
    }
    {{ val = a[0] + a[1]*x + … + a[n]*x^n }}
    return val;
}
```

UW CSE 331 Spring 2022 36
More Reasoning

Let’s check that this method is correct.

```java
/** Return the value of this IntPoly at point x */
public int valueAt(int x) {
    int val = a[0];
    int xk = 1;
    int k = 0;
    int n = a.length - 1;
    {{ inv: xk = x^k && val = a[0] + a[1]*x + ... + a[k]*x^k }}
    while (k != n) {
        {{ inv && k != n }}
        xk = xk * x;
        {{ xk = x^(k+1) && val = a[0] + a[1]*x + ... + a[k]*x^k && k != n }}
        val = val + a[k+1]*xk;
        {{ xk = x^(k+1) && val = a[0] + a[1]*x + ... + a[k+1]*x^(k+1) && k != n }}
        k = k + 1;
        {{ ? }}
    }
    {{ val = a[0] + a[1]*x + ... + a[n]*x^n }}
    return val;
}
```
More Reasoning

Let’s check that this method is correct.

```java
/** Return the value of this IntPoly at point x */
public int valueAt(int x) {
    int val = a[0];
    int xk = 1;
    int k = 0;
    int n = a.length - 1;
    {{ inv: xk = x^k && val = a[0] + a[1]*x + ... + a[k]*x^k }}
    while (k != n) {
        {{ inv && k != n }}
        xk = xk * x;
        {{ xk = x^(k+1) && val = a[0] + a[1]*x + ... + a[k]*x^k && k != n }}
        val = val + a[k+1]*xk;
        {{ xk = x^(k+1) && val = a[0] + a[1]*x + ... + a[k+1]*x^(k+1) && k != n }}
        k = k + 1;
        {{ inv && k-1 != n }} -> {{ inv }}
    }
    {{ val = a[0] + a[1]*x + ... + a[n]*x^n }}
    return val;
}
```

Do we reach the postcondition?
Good luck on the midterm!