- review reverse digits exercise
- introduce lab

Next week and homework
- next week: recursion
- homework by next week:
  - download, run and understand this week’s example programs
  - read Horstmann, sections 13.1 – 13.3

Lab
- first graded lab is assigned. See Canvas for due date
- read 'How labs are graded' in ‘Course information’

Review last week

- developed our first dynamic implementation, of the stack data structure, using a linked list

- emphasized an important technique where we draw pictures to design complex data structures

- used advanced Java features of an interface; generic data types and exceptions to develop a full, final implementation

- practiced applying this full final version, to the reverse digits programming exercise

Introduction to this week

- want to use the stack data structure to solve some problems

- review first the reverse digits programming exercise from last week

- introduce the first graded lab, applying stacks to the problem of adding very large numbers

Review reverse digits exercise
Objective: review this programming exercise on stacks, will help with the stack lab

<u>Used a stack to reverse digits</u>

- actually represented the integer as a `String`, to avoid overflow problems. So we need a stack of datatype `Character` to do the reverse:

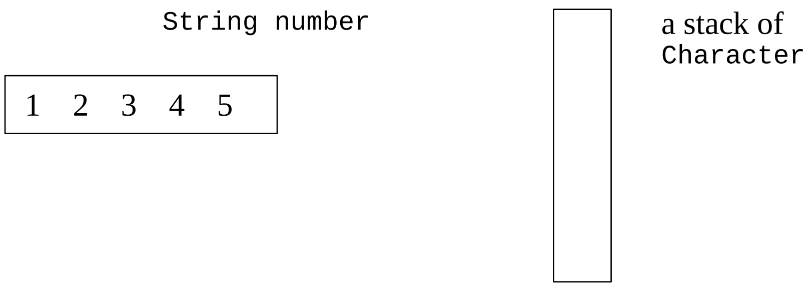

Figure 1 use a string and a stack of characters

– the reversing algorithm in pseudocode is something like:

```
loop for the digits in the number
   push digit to stack
while (!stack is empty)
   pop stack
```

– so reversing the digits would look something like:

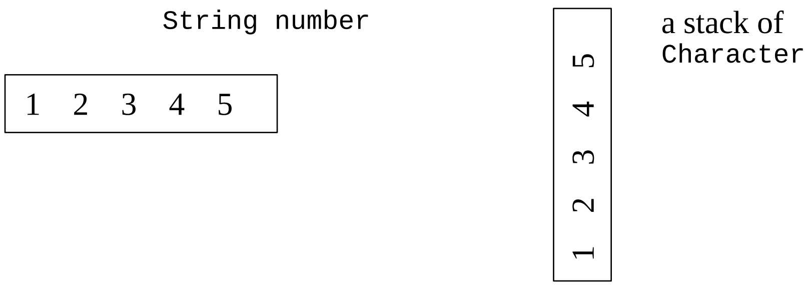

Figure 2 reversing, with all the digits pushed to the stack

– all the digits have been pushed to the stack

– now pop the stack, to get digits in reverse order

<u>All the action is in `Tester::main()`</u>

- the full implementation of the dynamic stack is given to us. We just write code in `main()` to use it. So something like:

```
        String number = "12345";
        StackInterface<Character> s = new LinkedStack<Character>();
```

  – see here how to create the stack of data type `Character`. The data type required is passed like a parameter to our genric `StackInterface` class. Cool

- first push digits to the stack:

```
     int len = number.length();
     for (int i = 0; i < len; ++i)
         s.push(number.charAt(i));
```

  – traverses all the digits in the number string, left to right

  – pushes each to the stack

- now pop the stack and the digits are reversed:

```
        while (!s.isEmpty())
            System.out.print(s.pop());
```

  – is as simple as possible

<u>Summary</u>

- this will help with the stack lab assigned this week

Introduce lab
Objective: introduce the first graded lab, which uses stacks to add large numbers

- we can use stacks to safely add integer values that overflow the `int` data type
  - e.g. in Java, the maximum possible `int` value `Integer.MAX_VALUE` is:

    2147483647
  - so any `int` addition larger than this will overflow and fail

Using stacks to add large numbers safely

- will actually represent the large integers to be added as strings of characters, to avoid any overflow problems here. Then will use three(!) stacks of datatype `Integer` to do the addition safely:

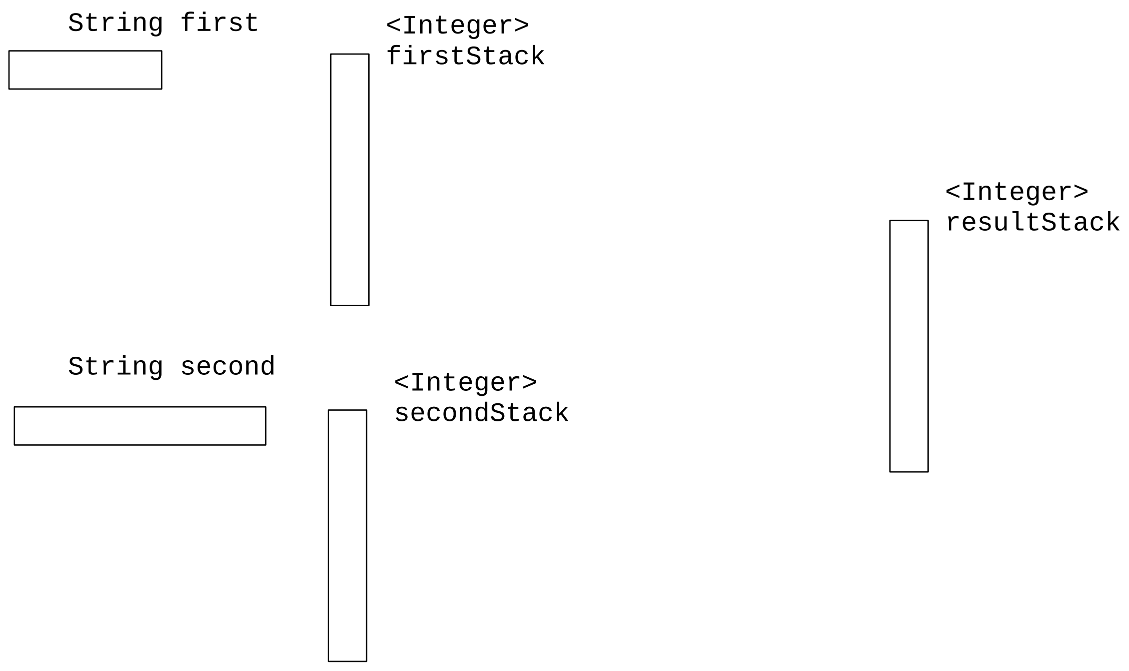

Figure 3 use two strings and three stacks

- the idea is to add each pair of digits from the numbers, from right to left
  - the units part of this total is pushed to the results stack
  - the tens part is the carry digit, added to the next pair

- addition algorithm in pseudocode is something like:

```
//push digits from numbers into stacks in appropriate order
loop for the digits in the first number
   push digit to first stack
loop for the digits in the second number
   push digit to second stack

//pop stacks, add digits and push result to result stack.  Careful with carry
while (!stacks are empty)
   pop digits from stacks and add
   push units part to result stack
   tens part is the carry to be added in next iteration

//print the result from the result stack
while (!result stack is empty)
   pop result stack
```

<u>Example: so adding numbers would look something like this</u>

- push digits from numbers into stacks in appropriate order
  - e.g., if:

```
String first = “543”;
String second = “45678”;
```

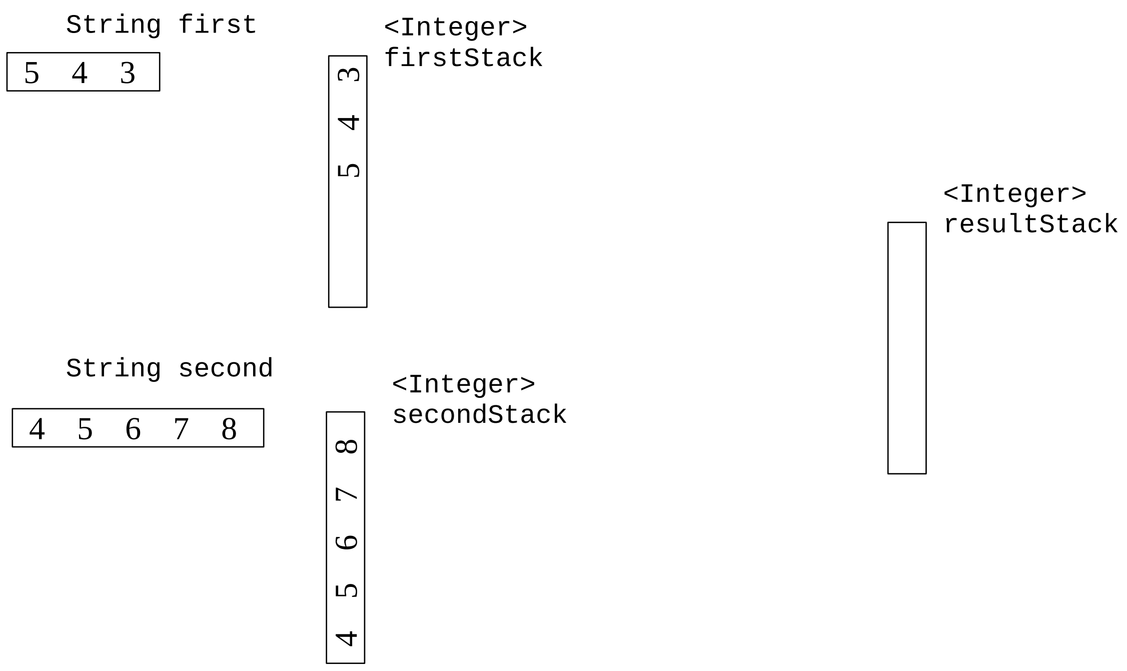

Figure 4 push the digits to the stacks

- pop stacks, add digits and push units for result to result stack. Careful with carry

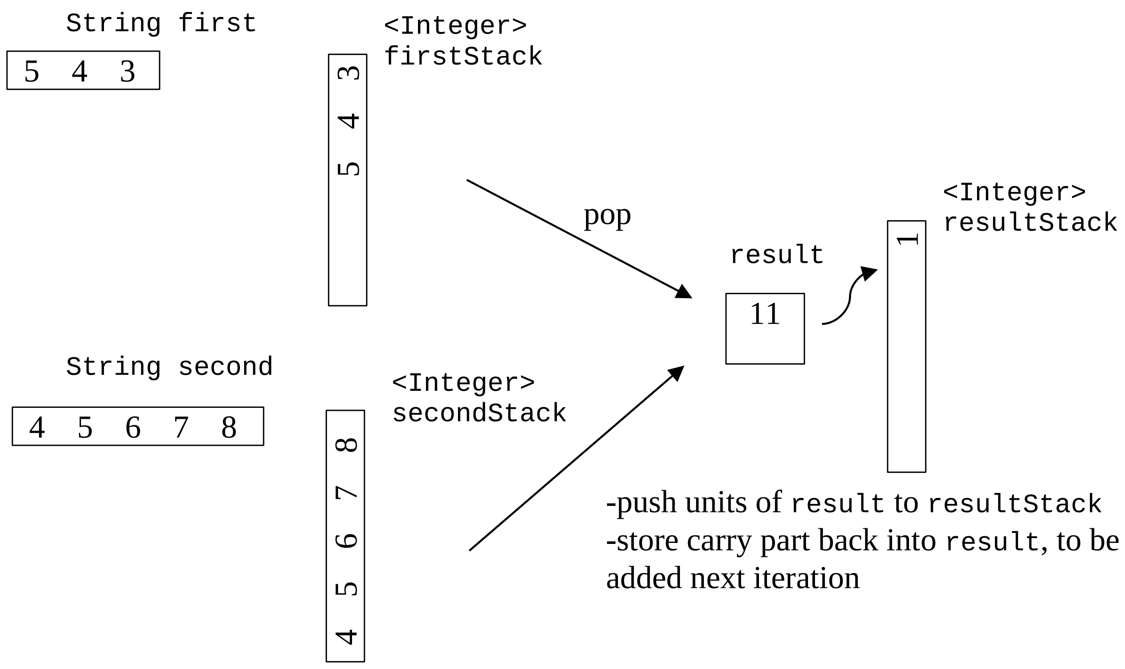

Figure 5 push first digit into the result stack

  – to find units from `result` is `result % 10` e.g. 11 % 10 above gives unit digit 1, pushed to `resultStack`

  – the carry digit is `result / 10` e.g. 11 / 10 above gives a carry of 1. Put this back into `result`, to be added to the next pair of digits, 4 and 7 here

  – e.g. 1 + 4 + 7 gives 12. Units 2 is pushed to `resultStack`, Carry 1 is put into `result`, to be added to the next pair 5 and 6

  – and so on

- be careful to add all of the digits from both of the numbers. There are three different possibilities:

  – first number is longer

  – second number is longer

  – same length

  – careful always to add any last carry correctly

- finally, print the result from the result stack. Easy

<u>Summary</u>

- this addition using stacks will be the first graded lab, assigned this week

Next week and homework

- next week: recursion

- homework by next week:

  - download, run and understand this week’s example programs

  - read Horstmann, sections 13.1 – 13.3

Lab

- first graded lab is assigned. See Canvas for due date

- read 'How labs are graded' in ‘Course information’