Java Recursion

Slides provided by the University of Washington Computer Science & Engineering department.

Adapted from slides by Marty Stepp, Stuart Reges & Allison Obourn.

Recursion

- **recursion**: The definition of an operation in terms of itself
  - Solving a problem using recursion depends on solving smaller occurrences of the same problem.
- **recursive programming:** Writing methods that call themselves to solve problems recursively
  - An equally powerful substitute for iteration (like for-loops)
  - Particularly well-suited for solving certain types of problems

Recursion and cases

- Every recursive algorithm involves at least 2 cases:
  - **base case:** A simple occurrence that can be answered directly
  - **recursive case:** A more complex occurrence of the problem that cannot be directly answered, but can instead be described in terms of smaller occurrences of the same problem
- Some recursive algorithms have more than one base or recursive case, but all have at least one of each.
- A crucial part of recursive programming is identifying these cases.

Recursion in Java

- Consider the following method to print a line of * characters:

```
// Prints a line containing the given number of stars.
// Precondition: n >= 0
public static void printStars(int n) {
  for (int i = 0; i < n; i++) {
    System.out.print("*");
  }
  System.out.println(); // end the line of output
}
```

- Write a recursive version of this method that calls itself.
  - Solve the problem without using any loops.
  - Hint: your solution should print just one star at a time.

Recursion in Java

- Our recursive solution is split into two cases:
  - **base case:** print 0 stars
  - **recursive case**: print more than 0 stars
- In Java:

```
public static void printStars(int n) {
  if (n == 0) {
    // base case; just end the line of output
    System.out.println();
  } else {
    // recursive case; print one zero stars
    System.out.print("*");
    printStars(n - 1);
  }
}
```

Recursive tracing

- Consider the following recursive method:

```
public static int mystery(int n) {
  if (n < 10) {
    return n;
  } else {
    int a = n / 10;
    int b = n % 10;
    return mystery(a + b);
  }
}
```

- What is the result of the following call?

```
mystery(648)
```

A recursive trace

```
mystery(648):

 -  int a = 648 / 10;            // 64
 -  int b = 648 % 10;            //  8
 -  return mystery(a + b);       // mystery(72)

    mystery(72):

     -  int a = 72 / 10;         // 7
     -  int b = 72 % 10;         // 2
     -  return mystery(a + b);    // mystery(9)

        mystery(9):

          -  return 9;
```

Recursive tracing 2

- Consider the following recursive method:

```
public static int mystery(int n) {
  if (n < 10) {
    return (10 * n) + n;
  } else {
    int a = mystery(n / 10);
    int b = mystery(n % 10);
    return (100 * a) + b;
  }
}
```

- What is the result of the following call?

```
mystery(348)
```

A recursive trace 2

```
mystery(348):

- int a = mystery(34);
    - int a = mystery(3);
        - return (10 * 3) + 3;        // 33
    - int b = mystery(4);
        - return (10 * 4) + 4;        // 44
    - return (100 * 33) + 44;         // 3344
- int b = mystery(8);
    - return (10 * 8) + 8;            // 88
- return (100 * 3344) + 88;           // 334488
```