# Building Java Programs

Chapter 5
Lecture 5-2: Random Numbers; procedural design

**reading: 5.1, 5.6, 4.5**
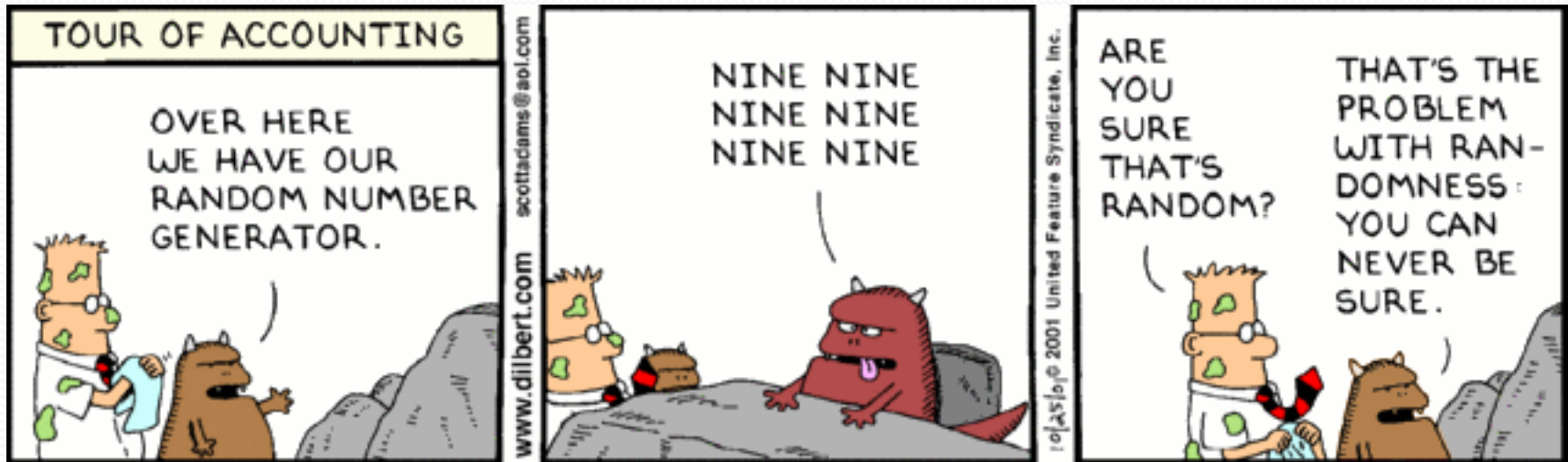
http://xkcd.com/221/

2

# The `Random` class

- A `Random` object generates *pseudo*-random numbers.
  - Class `Random` is found in the `java.util` package.
    ```
    import java.util.*;
    ```

| Method name | Description |
|---|---|
| `nextInt()` | returns a random integer |
| `nextInt(`**max**`)` | returns a random integer in the range [0, *max*) in other words, 0 to *max*-1 inclusive |
| `nextDouble()` | returns a random real number in the range [0.0, 1.0) |

  - Example:
    ```
    Random rand = new Random();
    int randomNumber = rand.nextInt(10);    // 0-9
    ```

# Generating random numbers

- Common usage: to get a random number from 1 to *N*

  ```
  int n = rand.nextInt(20) + 1;    // 1-20 inclusive
  ```

- To get a number in arbitrary range [*min*, *max*] inclusive:

  **name**.nextInt(**size of range**) + **min**

  - Where **size of range** is (**max** - **min** + 1)

  - Example: A random integer between 4 and 10 inclusive:

    ```
    int n = rand.nextInt(7) + 4;
    ```

# Random questions

- Given the following declaration, how would you get:
  ```
  Random rand = new Random();
  ```

  - A random number between 1 and 47 inclusive?
    ```
    int random1 = rand.nextInt(47) + 1;
    ```

  - A random number between 23 and 30 inclusive?
    ```
    int random2 = rand.nextInt(8) + 23;
    ```

  - A random even number between 4 and 12 inclusive?
    ```
    int random3 = rand.nextInt(5) * 2 + 4;
    ```

# Random and other types

- `nextDouble` method returns a `double` between [0.0, 1.0)

  - Example: Get a random GPA value between [1.5, 4.0):
    ```
    double randomGpa = rand.nextDouble() * 2.5 + 1.5;
    ```

- Any set of possible values can be mapped to integers
  - code to randomly play Rock-Paper-Scissors:
    ```
    int r = rand.nextInt(3);
    if (r == 0) {
        System.out.println("Rock");
    } else if (r == 1) {
        System.out.println("Paper");
    } else {   // r == 2
        System.out.println("Scissors");
    }
    ```

# Random question

- Write a program that simulates rolling two 6-sided dice until their combined result comes up as 7.

```
2 + 4 = 6
3 + 5 = 8
5 + 6 = 11
1 + 1 = 2
4 + 3 = 7
You won after 5 tries!
```

# Random answer

```java
// Rolls two dice until a sum of 7 is reached.
import java.util.*;

public class Dice {
    public static void main(String[] args) {
        Random rand = new Random();
        int tries = 0;

        int sum = 0;  // anything but 7 to start the loop
        while (sum != 7) {
            // roll the dice once
            int roll1 = rand.nextInt(6) + 1;
            int roll2 = rand.nextInt(6) + 1;
            sum = roll1 + roll2;
            System.out.println(roll1 + " + " + roll2 + " = " + sum);
            tries++;
        }

        System.out.println("You won after " + tries + " tries!");
    }
}
```

# Random question

- Write a program that plays an adding game.
  - Ask user to solve random adding problems with 2-5 numbers in the range from 1 - 10.
  - The user gets 1 point for a correct answer, 0 for incorrect.
  - The program stops after 3 incorrect answers.

```
4 + 10 + 3 + 10 = 27
9 + 2 = 11
8 + 6 + 7 + 9 = 25
Wrong! The answer was 30
5 + 9 = 13
Wrong! The answer was 14
4 + 9 + 9 = 22
3 + 1 + 7 + 2 = 13
4 + 2 + 10 + 9 + 7 = 42
Wrong! The answer was 32
You earned 4 total points
```

# Random answer

```java
// Asks the user to do adding problems and scores them.
import java.util.*;

public class AddingGame {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        Random rand = new Random();

        // play until user gets 3 wrong
        int points = 0;
        int wrong = 0;
        while (wrong < 3) {
            int result = play(console, rand);    // play one game
            if (result == 0) {
                wrong++;
            } else {
                points++;
            }
        }

        System.out.println("You earned " + points + " total points.");
    }
```

# Random answer 2

```
...

// Builds one addition problem and presents it to the user.
// Returns 1 point if you get it right, 0 if wrong.
public static int play(Scanner console, Random rand) {
    // print the operands being added, and sum them
    int operands = rand.nextInt(4) + 2;
    int sum = rand.nextInt(10) + 1;
    System.out.print(sum);

    for (int i = 2; i <= operands; i++) {
        int n = rand.nextInt(10) + 1;
        sum += n;
        System.out.print(" + " + n);
    }
    System.out.print(" = ");

    // read user's guess and report whether it was correct
    int guess = console.nextInt();
    if (guess == sum) {
        return 1;
    } else {
        System.out.println("Wrong! The answer was " + total);
        return 0;
    }
}
}
```

# Procedural design

**reading: 4.5**

# Recall: BMI program

Formula for body mass index (BMI):

$$BMI = \frac{weight}{height^2} \times 703$$

| BMI | Weight class |
|---|---|
| below 18.5 | underweight |
| 18.5 - 24.9 | normal |
| 25.0 - 29.9 | overweight |
| 30.0 and up | obese |

- Write a program that produces output like the following:

```
This program reads data for two people and
computes their body mass index (BMI).

Enter next person's information:
height (in inches)? 70.0
weight (in pounds)? 194.25

Enter next person's information:
height (in inches)? 62.5
weight (in pounds)? 130.5

Person 1 BMI = 27.868928571428572
overweight
Person 2 BMI = 23.485824
normal
Difference = 4.383104571428515
```
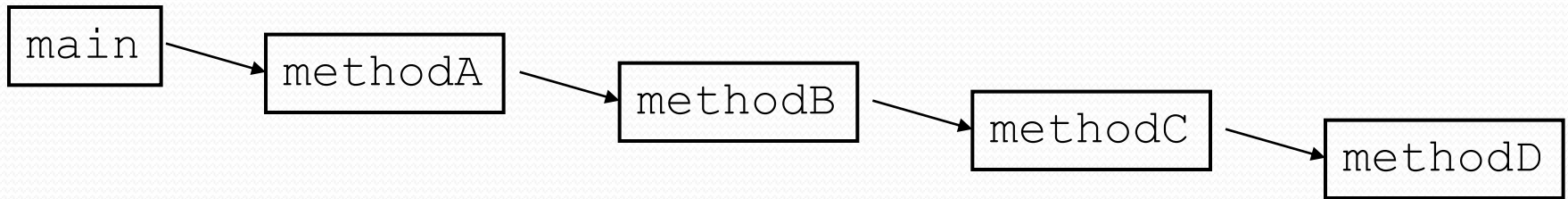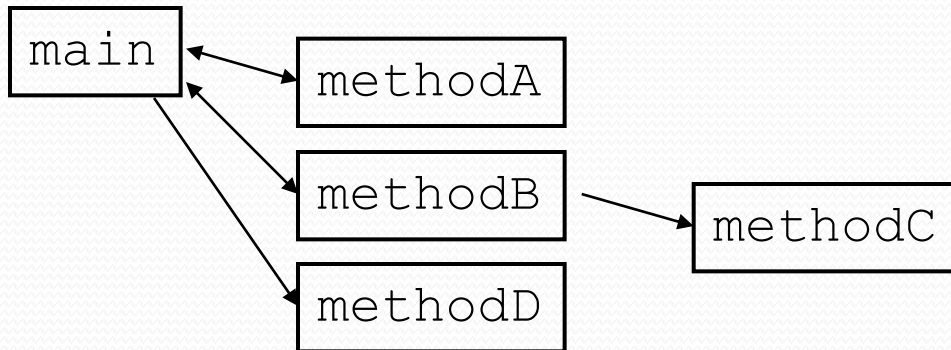
# "Chaining"

- `main` should be a concise summary of your program.
  - It is bad if each method calls the next without ever returning (we call this *chaining*):

```
main → methodA → methodB → methodC → methodD
```

- A better structure has `main` make most of the calls.
  - Methods must return values to `main` to be passed on later.

```
main ↔ methodA
main ↔ methodB → methodC
main ↔ methodD
```

# Bad "chain" code

```java
public class BMI {
    public static void main(String[] args) {
        System.out.println("This program reads ... (etc.)");
        Scanner console = new Scanner(System.in);
        person(console);
    }

    public static void person(Scanner console) {
        System.out.println("Enter next person's information:");
        System.out.print("height (in inches)? ");
        double height = console.nextDouble();
        getWeight(console, height);
    }

    public static void getWeight(Scanner console, double height) {
        System.out.print("weight (in pounds)? ");
        double weight = console.nextDouble();
        computeBMI(console, height, weight);
    }

    public static void computeBMI(Scanner s, double h, double w) {
        ...
    }
}
```

# Procedural heuristics

1. Each method should have a clear set of responsibilities.

2. No method should do too large a share of the overall task.

3. Minimize coupling and dependencies between methods.

4. The main method should read as a concise summary of the overall set of tasks performed by the program.

5. Data should be declared/used at the lowest level possible.