**Question**

- Why should we define and use methods?

**Why define methods**

- **Abstraction**
- Can call a method from multiple locations, so common code does not have to be repeated
  - Improves program structure and makes it easier to understand
  - Easier to debug
    - Only 1 place to get wrong
    - Only 1 place to fix

**Repeated code vs Method**

- **This?**
  ```java
  if (a < b)
      x = a;
  else
      x = b;
  ...
  if (j < k)
      y = j;
  else
      y = k;
  ```
- **Or this?**
  ```java
  x = min(a, b);
  ...
  y = min(j, k);
  ```

**More about return**

Control returns immediately from a method when a return is executed.

```java
static void printRoot(double x) {
    if (x < 0) {
        System.out.println("Error - negative number");
        return;
    }
    System.out.println("The square root of " + x);
}
```

```java
// Min2.java: return expression in a method
class Min2 {
    public static void main(String[] args) {
        int j = 78, k = 90, m;
        System.out.println("Minimum of two integers Test:");
        m = min(j, k);
        System.out.println("The minimum is "+ m);
    }
    static int min(int a, int b) {
        if (a < b)
            return a;
        else
            return b;
    }
}
```

**Return required**

- A method that returns a value must have at least one return statement

```java
class MissingReturn {
    public static void main(String[] args) {
        int myValue;
        myValue = getValue();
    }
    static int getValue() {
        System.out.println("Hi");
    }
    // This method should return a value
    static int getValue() {
        System.out.println("Fail");
    }
}
```
Return type

- The type of the expression in a return statement must be the same as the method type

```java
class BadReturn {
    public static void main( String[] args ) {
        int myValue;
        myValue = getValue();
    }
    // This method should return an int
    static int getValue() {
        return 2.1;
    }
}
```

Scope

- The `scope` of a variable is the range of statements that can access the variable
  - The scope of a variable extends from its declaration statement until the end of the block in which it is declared
    - This includes all inner blocks
  - The scope of a formal parameter is the entire method

```java
class Scope {
    public static void main( String[] args ) {
        int a = 10, b = 20, m;
        m = max();
        System.out.println("The max is " + m);
    }
    static int max() {
        if ( a > b )      // can't use a or b here
            return a;
        else
            return b;
    }
}
```

```java
for (int i; i < 10; i++) {
    System.out.print(i);
    System.out.println(" + Math.sqrt(i));
}
System.out.println("Can't access i here");
```

```java
for (int i; i < 10; i++) {
    System.out.print(i);
    System.out.print(" + Math.sqrt(i));
}
System.out.println("Can't access i here");
```

```java
What is the Scope?

// Min2.java: return expression in a method
class Min2 {
    public static void main(String[] args) {
        int j = 78, k = 90;
        int m;
        System.out.println("Minimum of two integers Test:");
        m = min(j, k);
        System.out.println("The minimum is "+ m);
    }
    static int min(int a, int b) {
        if (a < b)
            return a;
        else
            return b;
    }
}
```
Parameters: Pass by Value

- When you call a method, the value of each actual parameter is calculated and assigned to the formal parameter.
- Changes made to the value of the formal parameter are effective only in the method
- See MethodExample8.java

Recall: Software Life Cycle

- Requirements Analysis and Definition
- Design
- Implementation
- Testing
- Maintenance

Interest Rate Problem: Details

- Write a program that
  1. Gets the initial amount, the interest rate, and the target amount from the user
  2. Prints a nice header
  3. While the money in the bank is less than the target amount
     1. Calculates the interest amount for the year
     2. Displays the year number, the amount at the beginning of the year, the earned interest, and the total amount at the end of the year, like this:
        1  $100    $7    $107
  4. Prints a message "It will take you n years to get from $start to $end at x% interest".

Interest Rate Problem: Requirements

- What are the inputs?
  - What type is each input?
  - Do we need input validation?
- What are the outputs?
- Are there any constants?
- Do we need more details about the requirements?

Interest Rate Problem: Design

- Top Down design
- What's our top-level psuedocode?
**Interest Rate Problem: Design**

- Top Down design
- What's our top-level pseudocode?
  1. Get startAmount, interestRate, targetAmount from user
  2. year = 1, currentAmount = startAmount
  3. print header
  4. while currentAmount is less than targetAmount
     1. Calculate interest and newAmount
     2. print year, currentAmount, interest, newAmount
     3. currentAmount = newAmount
     4. year++
  5. print end message

**Interest Rate Problem: Revision**

- Revision – simplify the top level
- What's our top-level pseudocode?
  1. Get startAmount, interestRate, targetAmount from user
  2. print header
  3. year = numYearsToTarget( start, rate, target )
  4. print end message ( start, target, rate, year )

**NumYearsToTarget pseudocode**

1. year = 1
2. while ( start < target ) {
   1. interest = start * rate
   2. newAmount = start + interest
   3. print year, start, interest, newAmount
   4. start = newAmount
   5. year = year + 1
3. return year