An important topic: **preconditions** and **postconditions**.

They are one way of specifying what a method accomplishes.

Frequently a programmer must communicate precisely what a method accomplishes, without any indication of how the method does its work.

*Can you think of a situation where this would occur?*
Example

- You are the head of a programming team and you want one of your programmers to write a method for part of a project.

**HERE ARE THE REQUIREMENTS FOR A METHOD THAT I WANT YOU TO WRITE.**

**I DON'T CARE HOW THE METHOD WORKS, AS LONG AS THESE REQUIREMENTS ARE MET.**

What are Preconditions and Postconditions?

- One way to specify such requirements is with a pair of statements about the method.
- The **precondition** statement indicates what must be true before the method is called.
- The **postcondition** statement indicates what will be true when the method finishes its work.
Example

// Precondition: x >= 0.
// Postcondition: The square root of x has
// been written to the standard output.
public void writeSqrt( double x)

...
Example

// Precondition: x >= 0.
// Postcondition: The square root of x has
// been written to the standard output.
public void writeSqrt( double x)

- In this example, the precondition
  requires that
    x >= 0
  be true whenever the method is called.

Example

Which of these method calls meet the precondition?

writeSqrt( -10 );
writeSqrt( 0 );
writeSqrt( 5.6 );
Example

*Which of these method calls meet the precondition?*

```java
writeSqrt(-10);
writeSqrt(0);
writeSqrt(5.6);
```

The second and third calls are fine, since the argument is greater than or equal to zero.

Example

*Which of these method calls meet the precondition?*

```java
writeSqrt(-10);
writeSqrt(0);
writeSqrt(5.6);
```

But the first call violates the precondition, since the argument is less than zero.
Example

// Precondition: x >= 0.
// Postcondition: The square root of x has
// been written to the standard output.
public void writeSqrt( double x)

- The postcondition always indicates what work the method has accomplished. In this case, when the method returns the square root of x has been written.

Another Example

// Precondition: letter is an uppercase or lowercase letter (in the range 'A' ... 'Z' or 'a' ... 'z').
// Postcondition: The value returned by the method is true if letter is a vowel; otherwise the value returned by the method is false.
public boolean isVowel( char letter )
Another Example

What values will be returned by these method calls?

```java
isVowel('A');
isVowel('Z');
isVowel('?');
```

Nobody knows, because the precondition has been violated.
Another Example

What values will be returned by these method calls?

```java
isVowel('?');
```

Violating the precondition might even crash the program.

Always make sure the precondition is valid . . .

- The programmer who calls the method is responsible for ensuring that the precondition is valid when the method is called.
. . . so the postcondition becomes true at the method’s end.

- The programmer who writes the method counts on the precondition being valid, and ensures that the postcondition becomes true at the method’s end.

Then my method will execute, and when it is done, the postcondition will be true. I Guarantee it.

A Quiz

Suppose that you call a method, and you neglect to make sure that the precondition is valid. Who is responsible if this inadvertently causes a 40-day flood or other disaster?

- You
- The programmer who wrote that torrential method
- Noah
A Quiz

Suppose that you call a method, and you neglect to make sure that the precondition is valid. Who is responsible if this inadvertently causes a 40-day flood or other disaster?

You

The programmer who calls a method is responsible for ensuring that the precondition is valid.

On the other hand, careful programmers also follow these rules:

- When you write a method, you should make every effort to detect when a precondition has been violated.
- If you detect that a precondition has been violated, then print an error message and halt the program.
On the other hand, careful programmers also follow these rules:

- When you write a method, you should make every effort to detect when a precondition has been violated.
- If you detect that a precondition has been violated, then print an error message and halt the program...
- ...rather than causing a disaster.

Example

```java
// Precondition: x >= 0.
// Postcondition: The square root of x has been written to the standard output.
public void writeSqrt( double x)
{
  if (x < 0)
    throw new IllegalArgumentException("Negative x");

  ...

  Throwing an exception(described in Section 1.1) is useful.
```
Throw and catch exceptions

- Exception: an error message generated by a method to its calling method
- General usage:
  
  ```java
  if (condition is true)
      throw new ExeptionType ("error message");
  ```
- System and user defined exception types
- Informative error message
- Throwing an exception will halt the method and, without catching an exception, its calling method.

Example – temperature conversion

```java
/**
 * Convert a temperature from Celsius degrees to Fahrenheit degrees.
 * @param c a temperature in Celsius degrees
 * @precondition c >= -273.16.
 * @return the temperature c converted to Fahrenheit
 * @throws java.lang.IllegalArgumentException Indicates that c is less than the smallest Celsius temperature (-273.16).
 **/
public static double celsiusToFahrenheit(double c)
{
    final double MINIMUM_CELCIUS = -273.16;
    if (c < MINIMUM_CELCIUS)
        throw new java.lang.IllegalArgumentException("Argument " + c + " is too small.");
    return (9.0/5.0)*c + 32;
}
```
Example – temperature conversion

```
// part of the calling method

double celsius, fahrenheit;
celsius = -300;
try{
    fahrenheit = celsiusToFahrenheit(celsius);
    System.out.printf("%.2fC", celsius);
    System.out.printf("%.2fF\n", fahrenheit);
}
catch (IllegalArgumentException tooLowCelsius) {
    System.out.println("-300C is too small");
}
System.out.println("this will (not) print if there is (no) try/catch");
```

Summary

**Precondition**
- The programmer who calls a method ensures that the precondition is valid.
- The programmer who writes a method can bank on the precondition being true when the method begins execution.

**Postcondition**
- The programmer who writes a method ensures that the postcondition is true when the method finishes executing.
Advantages of Using Preconditions and Postconditions

- Succinctly describes the behavior of a method...
- ... without cluttering up your thinking with details of how the method works.
- At a later point, you may reimplement the method in a new way ...
- ... but programs (which only depend on the precondition/postcondition) will still work with no changes.

Presentation copyright 2012, Pearson Education
For use with Data Structures and Other Objects Using Java
by Michael Main.


Students and instructors who use Data Structures and Other Objects Using Java are welcome to use this presentation however they see fit, so long as this copyright notice remains intact.

THE END