CSE 1710

Lecture 6
Understanding Contracts

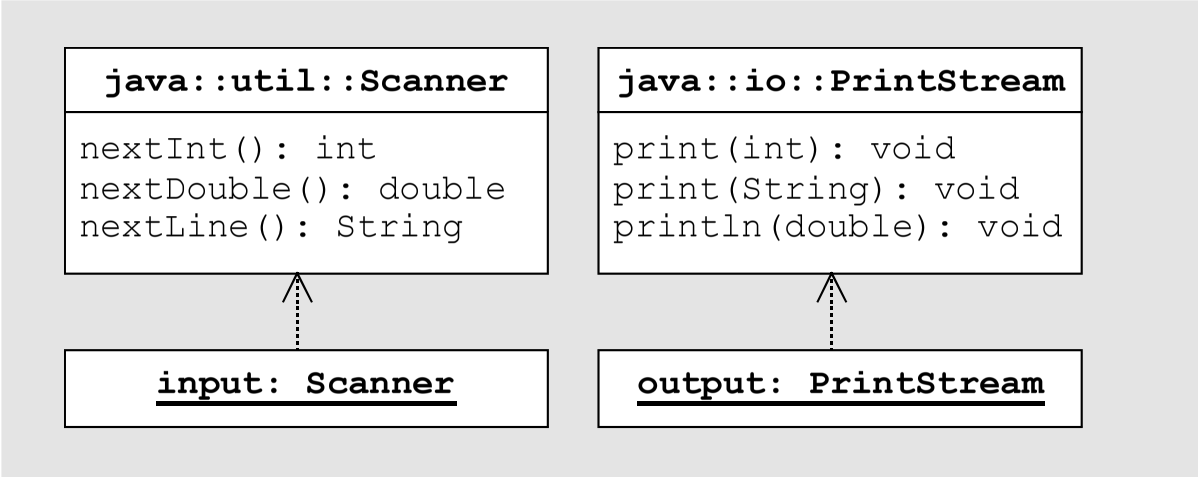
2.2.5 Ready-Made I/O Components

Keyboard Input:

```
Scanner input = new Scanner(System.in);
int width = input.nextInt();
```

Screen Output:

```
PrintStream output = System.out;
output.print(width);
```



Ready-Made I/O Components

Use this template as a starting point for all your programs in this course:

```
import java.util.Scanner;
import java.io.PrintStream;

public class Template
{
   public static void main(String[] args)
   {
      Scanner input = new Scanner(System.in);
      PrintStream output = System.out;
      ...
      //  use input.nextInt/Double for input
      //  use output.println/print for output
      ...
   }
}
```

2.3.1 Risk Mitigation by Early Exposure (RMBEE)

If you are not sure about something during software development, confront it as early as possible. Making changes later is more difficulty than doing so now.

Example:

the Java compiler turns a potential logic error (like assigning a real value to an int variable) to a compile-time error.

The risk of truncating the real value is exposed early.

2.3.2 Handling Constants

Replace all magic numbers (literals) in your program with finals.

Instead of:

```
width = width / 12;
```

Write:

```
final int INCH_PER_FOOT = 12;
width = width / INCH_PER_FOOT;
```

Advantages of finals versus literals:

The literal has a name and, thus, is self-documenting

The compiler prevents you from inadvertently change its value

2.3.2 Handling Constants

Blank finals:

Consider:

```
final int USER_SPECIFIED_DIVISOR;
// … what happens next?
```

2.3.3 Contracts – General Examples

`double squareRoot(double x)`

Returns the square root of the given argument.

Parameters

`x – an argument`

Pre

`x > 0`

Returns

The positive square root of x

Post

The return as stated under "Returns".

`double squareRoot(double x)`

Returns the square root of the given argument.

Parameters

`x – an argument`

Pre

`true`

Returns

The positive square root of x

Post

The return as stated under "Returns".

2.3.3 Who uses contracts?

Implementers (e.g., provider of `PrintStream` or `Date`)

[in development, early] conceptualize what the service should provide ***before*** actually implementing anything

[in development, testing] use contract as the basis for unit testing

[in "production"] stipulate terms and conditions of the use of the components

Clients (eg., you, developer of main method)

[in development, early] conceptualize what is needed, see whether the required services are provided by others

[in development, testing] diagnosis of problems

2.3.3 Contracts

Each method in a component comes with a contract that spells out the responsibilities of the client and the implementer.

The client must supply parameters that satisfy the precondition of the method.

The implementer must supply a return that satisfy the postcondition of the method.

Liability: if pre=false, the client is at fault, and if pre=true and post=false then the implementer is at fault. If pre=post=true then everything is OK.

Note: if a method has pre=true then its client does not have to ensure anything.

Contracts in Java

Methods in the Java standard library specify their pre and post as follows:

- **pre is always true unless stated otherwise**
- **post is specified under Returns and Throws**

Generic-style contract

`double squareRoot(double x)`

Returns the square root of the given argument.

Parameters

`x - an argument`

Pre

`true`

Returns

The positive square root of x

Post

The return as stated under "Returns".

Java-style contract

`double squareRoot(double x)`
Returns the square root of the given argument.

Parameters:
`x` – an argument.
Returns:
the positive square root of `x`.
Throws:
an exception if `x < 0`.