Lecture 8: Type Casting and Design

Last time:
1. Project
2. More assignment operators

Today:
1. Precedence and short-circuiting (from last lecture)
2. Type casting
3. Basics of program design
Project #1 Due Tomorrow (9/19) at 11 pm!

- The assignment is on the CMSC 131 web-site (click “Projects” link).
- It is due Tuesday, 9/19 at 11 pm
- The project is open
- Start now!
  - Read entire assignment from beginning to end before starting to code
  - Check out assignment now from CVS
  - Follow the instructions exactly, as much of grading is automated
Type Casting

Which of the following are legal?

- `int x = 3.5;`
  - **Illegal**: 3.5 is not an `int`
- `float x = 3;`
  - **Legal**: 3 is an `int`, which is also a `float`
- `long i = 3;`
  - **Legal**: 3 is an `int`, which is also a `long`
- `byte x = 155;`
  - **Illegal**: 155 is too big to be a `byte` (> 127)
- `double d = 3.14159F;`
  - **Legal**: 3.14159F is a `float`, which is also a `double`
What is “Type Casting”? 

- **Type casting**: automatic conversion of values from one type to another 
  e.g.  
  - `int` → `double`
  - `float` → `double`
  - `int` → `long`

- **Type casts can be**: 
  - **Implicit**: performed automatically 
  - **Explicit**: programmed by developer
Implicit Type Casting in Java

- Hierarchy of primitive numerical types:
  - double
  - float
  - long
  - int
  - short
  - byte

- Idea:
  - Higher types have more precision
  - Lower types are “subsets” of higher types

- Java only performs implicit upcasts (casting from lower to higher type)

  - e.g. int → double
  - float → double
  - int /→ byte
Explicit Type Casting in Java

- To explicitly cast a value to a type t, use \(<t>\) value
  ```java
  int x = (int)3.7;
  ```
  - Assigns value 3 to x
  - Reason: \((\text{int})\) operator converts double to int by truncation (chopping off decimal) when double is small enough
  - This is an example of **downcasting**
  ```java
  byte bt = (byte) 200;
  ```
  - Assigns value -56 to bt
  - Reason: \((\text{byte})\) operator “wraps around” values that are too big

- This sounds confusing
  - It is!
  - Rule of thumb:
    - Only use explicit casts when you know what the answer is likely to be
    - Otherwise (e.g. in “overflow” situations) write your own type-conversion routines (we’ll see how later this semester)
Mixed Expressions

- What is result of
  ```java
  float x = 3 / 4;
  ```
  - x assigned value `0.0F`
  - Why?
    - 3, 4 are ints
    - So integer / operation is used, yielding 0, before upcasting is performed

- To get floating point result, use explicit casting
  ```java
  float x = (float) 3 / (float) 4;
  ```
  - Assigns x the value `0.75F`

- Can also do following
  ```java
  float x = (float) 3 / 4;
  ```
  - Why?
    - `(float) 3` returns a value type float `(3.0F)`
    - 4 is an int
    - In this case, Java compiler uses upcasting on “lower” type (here, `int`) to obtain values in same type before computing operation
The Software Lifecycle

1. **Requirements**
   - What customers want
   - What you plan to do
   - Your program

2. **Design**
   - How you plan to do it

3. **Coding**
   - Create your program

4. **Testing**
   - Did you meet requirements?

5. **Deployment**
   - Delivery (documentation, etc.)

6. **Maintenance**
   - Bug fixes

7. **Evolution**
   - New versions

---

CMSC 131 Fall 2006
Rance Cleaveland
©2006 University of Maryland
So Far We’ve Seen…

- Coding
- Requirements (project assignment)
- Testing (submission testing)

What about design?
In the Real World, Requirements and Design Rule

- Getting requirements right is essential for successful projects
  - FBI electronic case file (junked after $180m)
  - IRS system upgrade in late 90s (junked after >$2bn)
  - FAA air-traffic control (false starts, >$10bn spent)
- Good design makes other parts of lifecycle easier
- In “the real world” coding typically < 30% of total project costs
Program Design

- There are many aspects to good design
  - Architecture
  - Modeling
  - Requirements decomposition
  - Pseudo-code
- In this class we will focus on latter
What Is “Pseudo-code”? 

- When developing a complex part of a program (an algorithm), one of the tools often useful is pseudo-code.
- It's not English, not programming language -- somewhere between.
- Captures the flow of the program without worrying about language-specific details.
Example:

- **Requirement**: email program that allows you to send a message either to one person, or to your whole address book

- **Pseudo-code**:

  ```
  prompt "Enter message: "
  input message
  prompt "Send to whole address book? "
  input answer
  if answer == "no"
    prompt "Enter recipient: "
    input recipient
    send message to recipient
  otherwise
    for each recipient, r, in address book
      send message to r
  ```
What Is Pseudo-Code? (cont.)

- NOT English
- NOT a program
- Something in-between
  - Captures the "logic" and "flow" of the algorithm
  - Note that pseudo-code could be translated into ANY programming language (not just Java)
- Good programming practice
  - Write pseudo-code first and keep it as your design
  - Include it as comments in your code to help you connect code to design
Testing

- Some testing is done by customer (acceptance testing)
  - E.g. testing we do on your projects!
  - You want to avoid errors surfacing during acceptance testing

- How to avoid errors during acceptance testing?
  - Test thoroughly before release
  - Cover all cases in code (if/else branches, etc.)
  - Identify “corner cases” (extreme values of inputs) and test with these

- We will study testing more later in semester