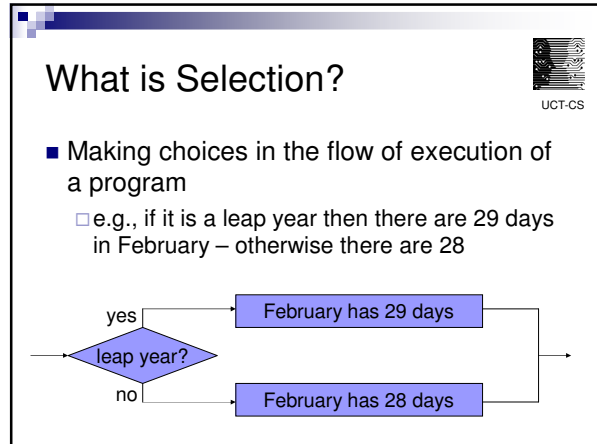


Selection

Hussein Suleman
UCT Dept of Computer Science
CS115 ~ 2004

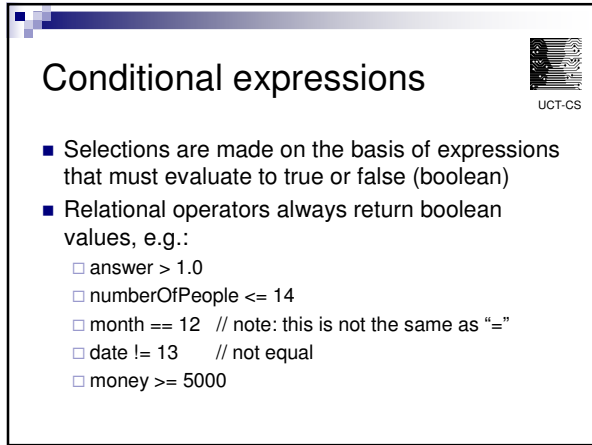


What is Selection?

- Making choices in the flow of execution of a program
 - e.g., if it is a leap year then there are 29 days in February – otherwise there are 28

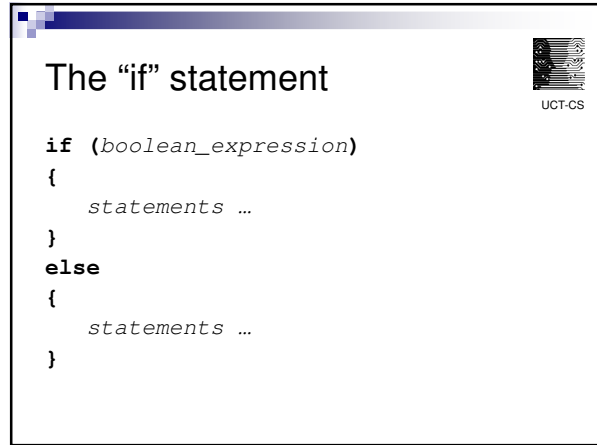
```

graph LR
    A[ ] --> B{leap year?}
    B -- yes --> C[February has 29 days]
    B -- no --> D[February has 28 days]
    C --> E[ ]
    D --> E
  
```



Conditional expressions

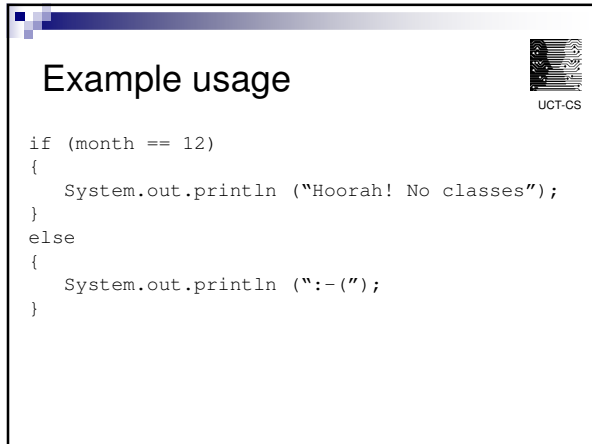
- Selections are made on the basis of expressions that must evaluate to true or false (boolean)
- Relational operators always return boolean values, e.g.:
 - `answer > 1.0`
 - `numberOfPeople <= 14`
 - `month == 12` // note: this is not the same as “=”
 - `date != 13` // not equal
 - `money >= 5000`



The “if” statement

```

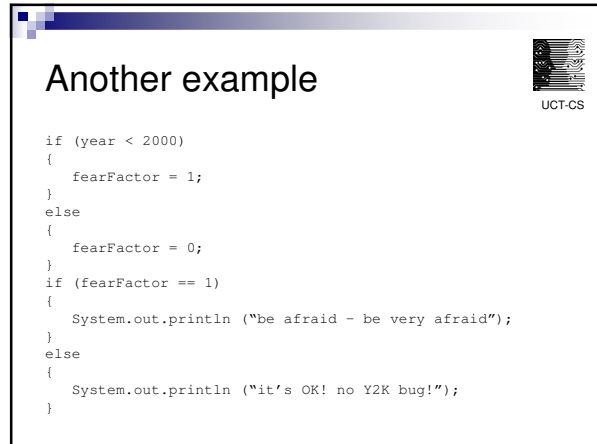
if (boolean_expression)
{
    statements ...
}
else
{
    statements ...
}
  
```



Example usage

```

if (month == 12)
{
    System.out.println ("Hoorah! No classes");
}
else
{
    System.out.println (":-(");
}
  
```



Another example

```

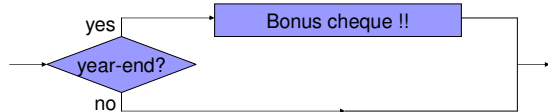
if (year < 2000)
{
    fearFactor = 1;
}
else
{
    fearFactor = 0;
}
if (fearFactor == 1)
{
    System.out.println ("be afraid - be very afraid");
}
else
{
    System.out.println ("it's OK! no Y2K bug!");
}
  
```

Shortcuts I



- No else part

```
if (numberOfStudents > 150)
{
    System.out.println ("Full!");
}
```



Shortcuts II



- Only one statement in block – can leave out the braces

```
if (numberOfStudents > 150)
    System.out.println ("Full!");
else
    System.out.println ("Not full!");
```

More Data Types



- char – stores a single character
 - char literals are enclosed in single quotes
 - e.g., char aLetter = 'a';
- boolean – stores only *true* or *false* values
 - e.g., boolean iLikeCSC115 = true;

```
if (iLikeCSC115)
{
    iEatWeetbix = true;
}
```

Issues with Strings



- You cannot compare two strings like other types of data
 - i.e., "Hello" == "Hello" may not work !
- Instead, use methods in String class
 - "Hello".compareTo("Hello") == 0
 - "Hello".equals ("Hello")
 - aString.compareTo ("somevalue") == 0
 - aString.equals ("somevalue")

Nested "if" statement



```
String password = Keyboard.readString();
if (password.equals (realPassword))
{
    if (name.equals ("admin"))
    {
        loggedIn = superPrivileges = true;
    }
}
else
{
    System.out.println ("Error");
}
```

Dangling Else



- Compiler cannot determine which "if" an "else" belongs to if there are no braces

```
String password = Keyboard.readString();
if (password.equals (realPassword))
    if (name.equals ("admin"))
        loggedIn = superPrivileges = true;
else
    System.out.println ("Error");
```

- Java matches else with *last unfinished if*
- Moral: Use shortcuts at your own risk – or don't !

Multiway selection



- Multiple conditions, each of which causes a different block of statements to execute
- Can be used where there are more than 2 options

```
if (condition1)
{
    statements ...
}
else
{
    if (condition2)
    {
        statements ...
    }
    else
    ...
}
```

“if” ladder



- Just a nicer way to write multiway selection

```
if (operation == 'a')
{
    answer = first + second;
}
else if (operation == 's')
{
    answer = first - second;
}
else if (operation == 'm')
{
    answer = first * second;
}
```

The “switch” statement



- Selects among different statements based on a single integer or character expression
- Each set of statements starts in “case” and ends in “break” because switch does not use {}s
 - break passes control to statement immediately after switch
- “default” applies if none of the cases match

Sample switch statement



```
switch (SouperSandwichOrder)
{
    case 1 : cheese = 1;
            break;
    case 2 : cheese = 1;
            chicken = 1;
            break;
    case 3 : cheese = 1;
            chicken = 1;
            chilli = 1;
            break;
    default : cheese = 1;
            break;
}
```

“break” optimisation



- If break is omitted, control continues to next statement in the switch

```
switch (SouperSandwichOrder)
{
    case 3 : chukka = 1;
    case 2 : tomato = 1;
    case 1 :
    default : cheese = 1;
}
```

Characters in “switch”



```
char Operation = Keyboard.readChar ("What to do?");
switch (Operation)
{
    case 'a' : answer = a + b;
            break;
    case 's' : answer = a - b;
            break;
    case 'm' : answer = a * b;
            break;
    case 'd' : if (b != 0)
            {
                answer = a / b;
                break;
            }
    default : answer = 0;
            System.out.println ("Error");
            break;
}
```

Boolean operators



Boolean Algebra	Java	Meaning
AND	&&	true if both parameters are true
OR		true if at least one parameter is true
NOT	!	true if parameter is false; false if parameter is true;

Operator precedence



- Now that we have seen how operators can be mixed, we need precedence rules for all operators
 - () (highest precedence – performed first)
 - !
 - * / %
 - + -
 - < <= > >=
 - == !=
 - &&
 - ||
 - = (lowest precedence – performed last)

Reversing expressions



- Use ! operator to reverse meaning of boolean expression, e.g.,

```
if (mark >= 0)
{
    // do nothing
}
else
    System.out.println ("Error");
```
- Instead, invert the condition

```
if (! (mark >= 0))
    System.out.println ("Error");
```
- Can we do better ?

Boolean operator example



```
boolean inClassroom, isRaining;
...
if (inClassroom && isRaining)
    System.out.println ("Lucky!");
...
if (! inClassroom && isRaining)
    System.out.println ("Wet and miserable!");
...
if (! isRaining && ! inClassroom)
    System.out.println ("Happy!");
```

Boolean expression example



```
int marks;
char symbol;
...
if (marks >= 75)
    symbol = 'A';
...
if (marks >= 65 && marks < 75)
    symbol = 'B';
...
if (marks < 0 || marks > 100)
{
    symbol = 'X';
    System.out.println ("Invalid mark!");
}
```

DeMorgan's Laws



- $\neg(A \ \&\& \ B) = \neg A \ || \ \neg B$
- $\neg(A \ || \ B) = \neg A \ \&\& \ \neg B$
- Invert the whole expression, the operators and the operands
 - $\neg(A \ \dots \ B) \rightarrow (A \ \dots \ B)$
 - $A \rightarrow \neg A$
 - $\&\& \rightarrow ||$
- Use this transformation to simplify expressions by removing "!"s wherever possible

Simplification



- Apply DeMorgan's Laws to simplify

```
(! (mark >= 0 && mark <= 100))  
(! (mark >= 0)) || (! (mark <= 100))  
(mark < 0 || mark > 100)
```
- Apply DeMorgan's Laws to simplify

```
! ( salary < 10000 || ! me.bigChief ())  
(! (salary < 10000)) && (!! me.bigChief ())  
salary >= 10000 && me.bigChief ()
```

Errors and testing



- Quick Poll
- In a typical hour spent programming, how many minutes do you spend fixing errors?

Errors



- What is an error?
 - When your program does not behave as intended or expected
- What is a bug?
 - "...a bug crept into my program ..."
- Debugging
 - the art of removing bugs



Types of Errors



- Compile-time Error
 - Discovered by Java when you hit "compile"
 - Improper use of Java language
 - e.g., `int x + 1;`
- Run-time Error
 - Program compiles but does not execute as expected
 - e.g., `int x=0, y = 15/x;`

Types of Errors II



- Logic Error
 - Program compiles and runs but produces incorrect results - because of a flaw in the algorithm or implementation of algorithm

```
int a = Keyboard.readInt();  
int b = Keyboard.readInt();  
int maximum;  
if (a < b) { maximum = a; }  
else { maximum = b; }
```

Testing Methods



- Programs must be thoroughly tested for all possible input/output values to make sure the programs behaves correctly
- But how do we test for all values of integers?

```
int a = Keyboard.readInt();  
if (a < 1 || a > 100)  
{ System.out.println ("Error"); }
```

Equivalence Classes



- Group input values into sets with similar expected behaviour and choose candidate values
 - e.g., -50, 50, 150
- Choose values at and on either side of boundaries (*boundary value analysis*)
 - e.g., 0, 1, 2, 99, 100, 101

Path Testing



- Create test cases to test every path of execution of the program at least once

```
int a = Keyboard.readInt();  
if (a < 1 || a > 100)  
{ System.out.println ("Error"); }
```

Path 1: a=35

Path 2: a=-5

Statement Coverage



- What if we had:

```
if (a < 25)  
{ System.out.println ("Error in a"); }  
if (b < 25)  
{ System.out.println ("Error in b"); }
```

- Rather than test all paths, test all statements at least once
 - e.g., (a,b) = (10, 10), (50, 50)

Quick Poll



- So, which of these is the best testing approach to use?
 1. Exhaustive testing of all values
 2. Equivalence classes and boundary values
 3. Path testing
 4. Statement coverage

Glass and Black Boxes



- If you can create your test cases based on only the problem specification, this is *black box testing*.
- If you have to look at the code, this is *glass box testing*.
- Which categories do these fall in:
 - Equivalence classes/boundary values
 - Path coverage
 - Statement coverage

Intro to Artificial Intelligence



- What is AI?
 - making machines appear to be intelligent
- Did you see the movie?
- Various approaches taken
 - complex algorithms
 - representing knowledge in a natural way
 - simulating the brain
 - simulating mother nature (e.g., evolution)

Neural Networks



- Can we simulate the brain by creating neuron “objects” and linking them together?
- How does the brain learn? and how does it recall information?
- Example:
 - EasyNN

The Turing Test



- If you put a machine and a computer behind a screen and communicate with them, can you figure out which is which?
- Famous example:
 - ELIZA