

CSC 221: Computer Programming I

Fall 2005

simple conditionals and expressions

- if statements, if-else
- increment/decrement, arithmetic assignments
- mixed expressions
- type casting
- operator precedence
- mixing numbers and Strings

1

Conditional execution

so far, all of the statements in methods have executed *unconditionally*

- when a method is called, the statements in the body are executed in sequence
- different parameter values may produce different results, but the steps are the same

many applications require *conditional execution*

- different parameter values may cause different statements to be executed

example: consider the `CashRegister` class

- previously, we assumed that method parameters were "reasonable"
 - i.e., user wouldn't pay or purchase a negative amount
 - user wouldn't check out unless payment amount \geq purchase amount
- to make this class more robust, we need to introduce conditional execution
 - i.e., only add to purchase/payment total if the amount is positive
 - only allow checkout if payment amount \geq purchase amount

2

If statements

in Java, an *if statement* allows for conditional execution

- i.e., can choose between 2 alternatives to execute

```
if (perform some test) {  
    Do the statements here if the test gave a true result  
}  
else {  
    Do the statements here if the test gave a false result  
}
```

```
public void recordPurchase(double amount)  
{  
    if (amount > 0) {  
        purchase = purchase + amount;  
    }  
    else {  
        System.out.println("ILLEGAL PURCHASE");  
    }  
}
```

if the test evaluates to true (amount > 0), then this statement is executed

otherwise (amount <= 0), then this statement is executed to alert the user

3

If statements (cont.)

you are not required to have an else case to an if statement

- if no else case exists and the test evaluates to false, nothing is done
- e.g., could have just done the following

```
public void recordPurchase(double amount)  
{  
    if (amount > 0) {  
        purchase = purchase + amount;  
    }  
}
```

but then no warning to user if a negative amount were entered (not as nice)

standard relational operators are provided for the if test

<	less than	>	greater than
<=	less than or equal to	>=	greater than or equal to
==	equal to	!=	not equal to

a comparison using a relational operator is known as a *Boolean expression*, since it evaluates to a *Boolean* (true or false) value

4

In-class exercises

update `recordPurchase` to display an error message if a negative amount

```
public void recordPurchase(double amount)
{
    if (amount > 0) {
        purchase = purchase + amount;
    }
    else {
        System.out.println("ILLEGAL PURCHASE");
    }
}
```

similarly, update `enterPayment`

5

In-class exercises

what changes should be made to `giveChange`?

```
public double giveChange()
{
    if ( _____ ) {
        double change = payment - purchase;
        purchase = 0;
        payment = 0;
        return change;
    }
    else {
    }
}
```

note: if a method has a non-void return type, every possible execution sequence must result in a return statement

- the Java compiler will complain otherwise

6

A further modification

suppose we wanted to add to the functionality of `CashRegister`

- get the number of items purchased so far
- get the average cost of purchased items

ADDITIONAL FIELDS?

CHANGES TO CONSTRUCTOR?

NEW METHODS?

7

Shorthand assignments

a variable that is used to keep track of how many times some event occurs is known as a *counter*

- a counter must be initialized to 0, then incremented each time the event occurs
- incrementing (or decrementing) a variable is such a common task that Java that Java provides a shorthand notation

`number++;` is equivalent to `number = number + 1;`

`number--;` is equivalent to `number = number - 1;`

other shorthand assignments can be used for updating variables

`number += 5;` \equiv `number = number + 5;` `number -= 1;` \equiv `number = number - 1;`

`number *= 2;` \equiv `number = number * 2;` `number /= 10;` \equiv `number = number / 10;`

```
public void recordPurchase(double amount)
{
    if (amount > 0) {
        purchase += amount;
    }
    else {
        System.out.println("ILLEGAL PURCHASE");
    }
}
```

8

Mixed expressions

note that when you had to calculate the average purchase amount, you divided the purchase total (`double`) with the number of purchases (`int`)

- mixed arithmetic expressions involving doubles and ints are acceptable
- in a mixed expression, the int value is automatically converted to a double and the result is a double

$2 + 3.5 \rightarrow 2.0 + 3.5 \rightarrow 5.5$

$120.00 / 4 \rightarrow 120.00 / 4.0 \rightarrow 30.0$

$5 / 2.0 \rightarrow 2.5$

- however, if you apply an operator to two ints, you always get an int result

$2 + 3 \rightarrow 5$

$120 / 4 \rightarrow 30$

$5 / 3 \rightarrow 2 ???$

CAREFUL: integer division throws away the fraction

9

Die revisited

extend the `Die` class to keep track of the *average* roll

- need a field to keep track of the total
- initialize the total in the constructors
- update the total on each roll
- compute the average by dividing the total with the number of rolls

```
public class Die
{
    private int numSides;
    private int numRolls;
    private int rollTotal;

    public Die() {
        numSides = 6;
        numRolls = 0;
        rollTotal = 0;
    }

    public Die(int sides) {
        numSides = sides;
        numRolls = 0;
        rollTotal = 0;
    }

    public int getNumberOfSides() {
        return numSides;
    }

    public int getNumberOfRolls() {
        return numRolls;
    }

    public double getAverageOfRolls() {
        return rollTotal/numRolls;
    }

    public int roll() {
        numRolls++;

        int currentRoll = (int)(Math.random()*getNumberOfSides() + 1);
        rollTotal += currentRoll;

        return currentRoll;
    }
}
```

PROBLEM: since `rollTotal` and `numRolls` are both ints, integer division will be used

- avg of 1 & 2 will be 1

UGLY SOLUTION: make `rollTotal` be a double

- kludgy! it really is an int

10

Type casting

a better solution is to keep `rollTotal` as an int, but *cast* it to a double when needed

- casting tells the compiler to convert from one compatible type to another
- general form:
`(NEW_TYPE)VALUE`
- if `rollTotal` is 3, the expression `(double)rollTotal` evaluates to 3.0

```
public class Die
{
    private int numSides;
    private int numRolls;
    private int rollTotal;

    public Die() {
        numSides = 6;
        numRolls = 0;
        rollTotal = 0;
    }

    public Die(int sides) {
        numSides = sides;
        numRolls = 0;
        rollTotal = 0;
    }

    public int getNumberOfSides() {
        return numSides;
    }

    public int getNumberOfRolls() {
        return numRolls;
    }

    public double getAverageOfRolls() {
        return (double)rollTotal/numRolls;
    }

    public int roll() {
        numRolls++;

        int currentRoll = (int)(Math.random()*getNumberOfSides() + 1);
        rollTotal += currentRoll;

        return currentRoll;
    }
}
```

you can cast in the other direction as well (from a double to an int)

- any fractional part is lost
- if `x` is 3.7 \rightarrow `(int)x` evaluates to 3

11

Complex expressions

how do you evaluate an expression like `1 + 2 * 3` and `8 / 4 / 2`

Java has rules that dictate the order in which evaluation takes place

- `*` and `/` have *higher precedence* than `+` and `-`, meaning that you evaluate the part involving `*` or `/` first

`1 + 2 * 3` \rightarrow `1 + (2 * 3)` \rightarrow `1 + 6` \rightarrow 7

- given operators of the same precedence, you evaluate from left to right

`8 / 4 / 2` \rightarrow `(8 / 4) / 2` \rightarrow `2 / 2` \rightarrow 1

`3 + 2 - 1` \rightarrow `(3 + 2) - 1` \rightarrow `5 - 1` \rightarrow 4

GOOD ADVICE: don't rely on these (sometimes tricky) rules

- place parentheses around sub-expressions to force the desired order

`(3 + 2) - 1`

`3 + (2 - 1)`

12

Mixing numbers and Strings

recall that the + operator can apply to Strings as well as numbers

- when + is applied to two numbers, it represents addition: $2 + 3 \rightarrow 5$
- when + is applied to two Strings, it represents concatenation: "foo" + "bar" \rightarrow "foobar"
- what happens when it is applied to a String and a number?

when this occurs, the number is automatically converted to a String (by placing it in quotes) and then concatenation occurs

```
x = 12;  
System.out.println("x = " + x);
```

- be very careful with complex mixed expressions

```
System.out.println("the sum is " + 5 + 2);
```

```
System.out.println(2 + 5 + " is the sum");
```

- again, use parentheses to force the desired order of evaluation

```
System.out.println("the sum is " + (5 + 2));
```

13