Introduction to Software Development (ISD)
Week 2

Autumn term 2012

londonknowledgelab

Aims of Week 2

- To learn about converting between different number types and between strings and numbers
- To learn about formatting numeric and string output
- To meet the `char` type
- To program decisions using the `if` statement
- To compare integers, floating-point numbers and strings
- To meet the `boolean` type
- To validate users' input to programs

Converting between *integers* and *floating point numbers* (numbers that may have a factional part)

- You can automatically use an integer whenever a double would be expected
- But going the other way, all fractional information would be lost. So the compiler disallows this kind of assignment e.g. it disallows

```
double balance = ...;
int dollars = balance;
```

- To force a type conversion such as this to happen, we can ***cast*** a variable of one type to another type e.g.

```
double balance = ...;
int dollars = (int) balance;
```

using this, the fractional part of `balance` is discarded N.B. it is truncated, *not* rounded

Rounding floating point numbers

- If we need to *round* a floating point number to the nearest integer, we can use the Math.round method:

```
double balance = ...;
long dollars = Math.round(balance);
```

- The type returned by Math.round is long not int because large floating point numbers don't fit into int. If we know that the result *does* fit into an int and does not require a long, we can use a cast:

```
double balance = ...;
int dollars = (int) Math.round(balance);
```

Unexpected errors can happen with casting!

```
public class RoundoffDemo
{
  public static void main(String[] args)
  {
    double price = 4.35;
    int pence = (int) (100 * price); // Should be 435
    System.out.println(pence);       // Prints 434!
  }
}
```

- The `double` value 4.35 is not stored exactly in the computer's memory (which is based on binary arithmetic i.e. base 2)
- Multiplying it by 100 gives 434.9999999999994
- Casting this to an `int`, the entire fractional part is thrown away!

Using round, instead:

```
public class RoundoffDemo
{
  public static void main(String[] args)
  {
    double price = 4.35;
    int pence = (int) Math.round(100 * price);
    System.out.println(pence);       // Prints 435
  }
}
```

- The `double` value 4.35 is not stored exactly in the computer's memory (which is based on binary arithmetic i.e. base 2)
- Multiplying it by 100 gives 434.99999999999994
- Rounding this to the nearest integer gives 435

Converting between strings and numbers

- If one of the arguments to the + operator is a String, the other argument is automatically forced into a string. This is useful for printing output e.g.

```
int total = 23456 ;
System.out.println("The total is " + total);
```

outputs:

```
The total is 23456
```

Converting between strings and numbers

- Going the other way, strings that contain only numbers can be converted to a number type by using the methods **`Integer.parseInt`** and **`Double.parseDouble`** e.g.

```
System.out.print("Please enter your age: ");
String input = in.next();
int age = Integer.parseInt(input);

System.out.print("Please enter your height in
  metres: ");
input = in.next();
double height = Double.parseDouble(input);
```

Formatting Output

- Outputting floating point values can sometimes look strange:

```
Balance is: 25157.78626
```

- To control the output appearance of variables, we can use the method System.out.printf :

```
double balance = 25157.78626;
System.out.printf("%.2f", balance);
```

outputs two decimal places:

```
25157.79
```

```
System.out.printf("%10.2f", balance);
```

outputs the number using a total of 10 spaces:

```
  25157.79
```

(two blank spaces, plus 8 more spaces for the number itself)

Formatting Output

- We can also include text inside the quotes:

```
double balance = 25157.78626;
System.out.printf("Balance is:%10.2f", balance);
```

outputs the floating point number as follows:

```
Balance is:  25157.79
```

-
```
int accountNo = 87651;
System.out.printf("Account number is:%8d", accountNo);
```

outputs the integer as follows:

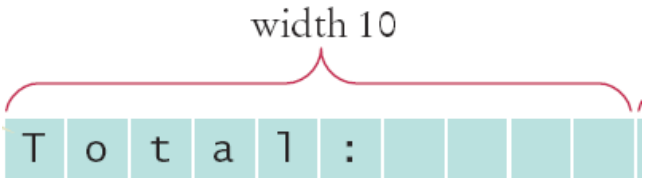
```
Account number is:   87651
```

Formatting Output

- We can use "%10s" to output a string using 10 spaces (right-justified)

- To left-justify a string, use the - "flag":

```
System.out.printf("%-10s", "Total:");
```



- We can print multiple values with a single call to printf:

```
System.out.printf("%-10s%10.2f", "Total:", price);
```

- And output a newline with '\n' :

```
System.out.printf("%-10s%10.2f\n", "Total:", price);
```

Strings and Characters

- We have already seen that Strings are sequences of characters
  - Characters have their own type in Java: `char`
  - Characters are encoded using numbers, in Unicode:
    - See the code chart in Appendix A of *Java for Everyone*
- We use single quotes around a value of type `char`:

```
char initial = 'A';
```

or equivalently, in Unicode:

```
char initial = '\u0041'; // hexadecimal 41
```

- And double quotes around a value of type `String`:

```
String initials = "AMP";
```

- The `charAt` method returns the `char` at a given position in a `String`:

```
char start = initials.charAt(0);
char last = initials.charAt(initials.length()-1);
```

Making decisions: the `if` statement

- The `if` statement allows a program to carry out different actions depending on the user's input or, more generally, the values of particular variables

- The two keywords of the `if` statement are:
  - `if`
  - `else`

  - If the condition specified in the `if` part is true, then that branch of the program is executed, otherwise the `else` branch is executed.

  - For example:

Example program

```
System.out.print("Please enter the price : ");
Scanner in = new Scanner(System.in);
double price  = in.nextDouble();
if (price > 50) {
    double discountRate = 0.15;
    double discountedPrice = (discountRate+1) * price;
    System.out.printf("%s%.2f\n","The price is ",discountedPrice);
}
else {
    double discountRate = 0.05;
    double discountedPrice = (discountRate+1) * price;
    System.out.printf("%s%.2f\n","The price is ",discountedPrice);
}
```

Using Braces

- Two alternatives to formatting `if` statements:
  - Braces lined up

```
if (price > 50)
{
   . . .
}
else {
   . . .
}
```

Braces not aligned (saves lines)

```
if (price > 50) {
   . . .
}
else {
   . . .
}
```

- Always use braces even though clauses only containing a single statement do not require them – makes programs easier to read and maintain

Always use indenting in your programs - makes code much easier to read and to understand

- Use <Tab> in your editor to indent a consistent number of spaces:

```
public class ElevatorSimulation
{
|  public static void main(String[] args)
|  {
|  |  int floor;
|  |  . . .
|  |  if (floor > 13)
|  |  {
|  |  |  floor--;
|  |  }
|  |  . . .
|  }
|  |  |  |
0  1  2  3    Indentation level
```

From:Java for Everyone by Cay Horstmann

Sometimes an `else` branch isn't needed:

```
if (floor > 13) {
   floor--;
}
System.out.println("Actual floor is " + floor);
```

- Be careful with `;' The following is legal Java code but probably not what the programmer intended:

```
if (floor > 13);
{
   floor--;
}
System.out.println("Actual floor is " + floor);
```

Avoid duplication of code in different branches:

```
System.out.print("Please enter the price : ");
Scanner in = new Scanner(System.in);
double price  = in.nextDouble();
if (price > 50) {
    double discountRate = 0.15;
    double discountedPrice = (discountRate+1) * price;
    System.out.printf("%s%.2f\n","The price is ",discountedPrice);
}
else {
    double discountRate = 0.05;
    double discountedPrice = (discountRate+1) * price;
    System.out.printf("%s%.2f\n","The price is ",discountedPrice);
}
```

But what is the problem with this:

```
System.out.print("Please enter the price : ");
Scanner in = new Scanner(System.in);
double price  = in.nextDouble();
if (price > 50) {
    double discountRate = 0.15;
}
else {
    double discountRate = 0.05;
}
double discountedPrice = (discountRate+1) * price;
System.out.printf("%s%.2f\n","The price is ",discountedPrice);
```

But what is the problem with this:

```
System.out.print("Please enter the price : ");
Scanner in = new Scanner(System.in);
double price  = in.nextDouble();
if (price > 50) {
    double discountRate = 0.15;
}
else {
    double discountRate = 0.05;
}
double discountedPrice = (discountRate+1) * price;
System.out.printf("%s%.2f\n","The price is ",discountedPrice);
```

- *Answer:* there is no variable discountRate within the `scope' of the last two statements i.e. within the scope of the statements in red

This is now correct:

```
System.out.print("Please enter the price : ");
Scanner in = new Scanner(System.in);
double price  = in.nextDouble();
double discountRate;
if (price > 50) {
    discountRate = 0.15;
}
else {
    discountRate = 0.05;
}
double discountedPrice = (discountRate+1) * price;
System.out.printf("%s%.2f\n","The price is ",discountedPrice);
```

Comparing numbers in `if` statements

Table 1 Relational Operators

Java	Math Notation	Description
>	>	Greater than
>=	≥	Greater than or equal
<	<	Less than
<=	≤	Less than or equal
==	=	Equal
!=	≠	Not equal

Operator Precedence

- The comparison operators have lower precedence than arithmetic operators i.e. arithmetic calculations are done before the comparison

- So

```
if (age < retirement – 5)
```

is equivalent to

```
if (age < (retirement – 5))
```

- Tip about parentheses: these must always balance! To check this, start counting from the left with 0. Add 1 for every “(” encountered and delete 1 for every “)” encountered. You need to end up with 0 again.

Comparing floating point numbers

- Rounding errors can lead to unexpected results:

```
double price = 4.35;
double pence = 100 * price;
if (pence == 435) {
    . . .
}
else . . .
```

- It is therefore often better to test if floating point numbers are within some (small) threshold:

```
final double EPSILON = 1E-5;
double price = 4.35;
double pence = 100 * price;
if (Math.abs(pence - 435) < EPSILON) {
    . . .
}
else . . .
```

Comparing Strings

- Do not use the **==** operator with Strings. This is because **==** compares the *locations* of two strings in memory, and not their actual *contents*

- Instead, to compare the contents of two strings, we need to use the **equals** method of the String class:

```
String string1 = . . . ; // a string variable
String string2 = . . . ; // another string variable
if (string1.equals(string2)) . . .
```

Comparing Strings

- Similarly, do not use >, < etc. with Strings

- The method **`compareTo`** is based on the *lexicographic ordering* of characters (see Appendix A of *Java for Everyone*) – this is similar to dictionary order except that uppercase letters come before lowercase, numbers come before letters etc.

`string1.compareTo(string2) < 0`

means that string1 precedes string2 lexicographically;

`string1.compareTo(string2) == 0`

means that string1 has the same contents as string2;

`string1.compareTo(string2) > 0`

means that string1 follows string2 lexicographically;

Multiple Alternatives in Programs

- What if we need more than two branches? Use **`else if`** :

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

Example

```
System.out.print("Please enter the mark : ");
Scanner in = new Scanner(System.in);
int mark  = in.nextInt();
if (mark >= 70) {
    System.out.println("Distinction");
}
else if (mark >= 60) {
    System.out.println("Merit");
}
else if (mark >= 50) {
    System.out.println("Pass");
}
else {
    System.out.println("Fail");
}
```

Choosing Test Cases to test your program

- Choose input values that:
  - test each branch
    - e.g. try previous program with values 45, 55, 65, 75
  - test “boundary values” and check these behave as you intend
    - e.g. try previous program with values 0, 50, 60, 70, 100

Example 2 – What's wrong with this?

```
System.out.print("Please enter the mark : ");
Scanner in = new Scanner(System.in);
int mark  = in.nextInt();
if (mark >= 70) {
    System.out.println("Distinction");
}
if (mark >= 60) {
    System.out.println("Merit");
}
if (mark >= 50) {
    System.out.println("Pass");
}
else {
    System.out.println("Fail");
}
```

Example 3 – What's wrong with this:

```
System.out.print("Please enter the mark : ");
Scanner in = new Scanner(System.in);
int mark  = in.nextInt();
if (mark >= 50) {
    System.out.println("Pass");
}
else if (mark >= 60) {
    System.out.println("Merit");
}
else if (mark >= 70) {
    System.out.println("Distinction");
}
else {
    System.out.println("Fail");
}
```

We can *nest* an if inside either branch of an if to make more complex decisions e.g.

```
/*  this code fragment reads in two times expressed in the 24-hour
    clock and prints out which one is earlier */

Scanner in = new Scanner(System.in);
System.out.println("Enter two times (24 hour clock): ");
String time1 = in.next();
String time2 = in.next();

int hour1 = Integer.parseInt(time1.substring(0, 2));
int hour2 = Integer.parseInt(time2.substring(0, 2));

int minute1 = Integer.parseInt(time1.substring(2));
int minute2 = Integer.parseInt(time2.substring(2));
```

```
if (hour1 < hour2)  {
     System.out.println (time1 + " comes first");
}
else if (hour1 == hour2)  {
     if (minute1 < minute2)   {
          System.out.println (time1 + " comes first”);
    }
    else if (minute1 == minute2)   {
          System.out.println (time1 + " and " + time2 + " are the same");
   }
    else   {
        System.out.println (time2 + " comes first");
    }
}
else   {
    System.out.println (time2 + " comes first");
}
```

The boolean type

- The **boolean** type has just two values:
  - **true** and **false**
- We can declare a variable of boolean type if we want to store the value of a condition and use it elsewhere in the program e.g.

```
boolean hasPassed = mark >= 50;
. . .
if (has Passed) . . .
```

- We can combine conditions and boolean variables by using the *boolean operators* **&&** and **||** and **!** :

&& is the *and* operator

|| is the *or* operator

! is the *not* operator

Using the boolean opeators

- Combining two conditions is often used in checking that a value falls into a required range
- Both sides of an *and* must be true for the result to be true e.g.

```
if (mark >= 0 && mark <= 100) {
    System.out.println ("Valid mark");
}
```

- At least one side of an *or* must be true for the result to be true e.g.

```
if (mark < 0 || mark > 100) {
    System.out.println ("Invalid mark");
}
```

- An important application for the if statement and the boolean operators is *validation of program input:*

```
Scanner in = new Scanner(System.in);
System.out.println("Input an integer: ");
boolean validInput = true;
int mark = 0;
if (! in.hasNextInt()) {
    validInput = false;
}
else {
    mark = in.nextInt();
    if (mark < 0 || mark > 100) {
        validInput = false;
    }
}
if (validInput) {
    . . .
}
else System.out.println(“Invalid input”);
```

Aims of Week 2

- To learn about converting between different number types and between strings and numbers
- To learn about formatting numeric and string output
- To meet the `char` type
- To make decisions in programs using the `if` statement
- To compare integers, floating-point numbers and strings
- To meet the `boolean` type
- To validate users' input to programs

Week 2 Homework

- Complete Lab Sheet 2 – not assessed. Solutions will be posted next week on the website.
- Finish reading Chapter 2 and read Chapter 3 of *Java for Everyone* and do the self-check questions
- Make sure you read How To 3.1 (Implementing an `if` statement)
- If you have time, do some of the review and programming exercises from Chapter 3 of *Java for Everyone*

Summary of Format Types

Table 8 Format Types

Code	Type	Example
d	Decimal integer	123
f	Fixed floating-point	12.30
e	Exponential floating-point	1.23e+1
g	General floating-point (exponential notation is used for very large or very small values)	12.3
s	String	Tax:

From: Java for Everyone by Cay Horstmann

Summary of Format Flags

Table 9 Format Flags

Flag	Meaning	Example
-	Left alignment	1.23 followed by spaces
0	Show leading zeroes	001.23
+	Show a plus sign for positive numbers	+1.23
(	Enclose negative numbers in parentheses	(1.23)
,	Show decimal separators	12,300
^	Convert letters to uppercase	1.23E+1

From: Java for Everyone by Cay Horstmann

Character Testing Methods

- The `Character` class has a number of useful methods that return a `boolean` value:

Table 5 Character Testing Methods

Method	Examples of Accepted Characters
`isDigit`	0, 1, 2
`isLetter`	A, B, C, a, b, c
`isUpperCase`	A, B, C
`isLowerCase`	a, b, c
`isWhiteSpace`	space, newline, tab

Relational Operator Use (1)

Table 2 Relational Operator Examples

Expression	Value	Commment
`3 <= 4`	`true`	3 is less than 4; <= tests for "less than or equal".
`3 =< 4`	**Error**	The "less than or equal" operator is <=, not =<. The "less than" symbol comes first.
`3 > 4`	`false`	> is the opposite of <=.
`4 < 4`	`false`	The left-hand side must be strictly smaller than the right-hand side.
`4 <= 4`	`true`	Both sides are equal; <= tests for "less than or equal".
`3 == 5 - 2`	`true`	== tests for equality.
`3 != 5 - 1`	`true`	!= tests for inequality. It is true that 3 is not 5 – 1.

Relational Operator Use (2)

Table 2 Relational Operator Examples

`3 = 6 / 2`	**Error**	Use == to test for equality.
`1.0 / 3.0 == 0.333333333`	`false`	Although the values are very close to one another, they are not exactly equal. See Common Error 3.2 on page 87.
`"10" > 5`	**Error**	You cannot compare a string to a number.
`"Tomato".substring(0, 3).equals("Tom")`	`true`	Always use the `equals` method to check whether two strings have the same contents.
`"Tomato".substring(0, 3) == ("Tom")`	`false`	Never use == to compare strings; it only checks whether the strings are stored in the same location. See Common Error 3.3 on page 88.

Boolean Operator Examples

Table 6 Boolean Operator Examples

Expression	Value	Comment
`0 < 200 && 200 < 100`	`false`	Only the first condition is true.
`0 < 200 || 200 < 100`	`true`	The first condition is true.
`0 < 200 || 100 < 200`	`true`	The `||` is not a test for "either-or". If both conditions are true, the result is true.
`0 < x && x < 100 || x == -1`	`(0 < x && x < 100) || x == -1`	The `&&` operator has a higher precedence than the `||` operator (see Appendix B).
`0 < x < 100`	`Error`	**Error:** This expression does not test whether `x` is between `0` and `100`. The expression `0 < x` is a Boolean value. You cannot compare a Boolean value with the integer `100`.

Boolean Operator Examples

Table 6 Boolean Operator Examples

`x && y > 0`	`Error`	**Error:** This expression does not test whether `x` and `y` are positive. The left hand side `x` of `&&` is an integer, the right hand side `y > 0` is a Boolean value. You cannot use `&&` with an integer argument.
`!(0 < 200)`	`false`	`0 < 200` is `true`, therefore its negation is `false`.
`frozen == true`	`frozen`	There is no need to compare a Boolean variable with `true`.
`frozen == false`	`!frozen`	It is clearer to use `!` than to compare with `false`.