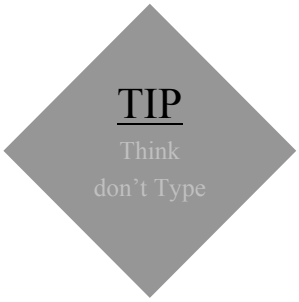
Problem Solving and Pseudocode

Think
don't Type



Overview

In Java, a large program is subdivided into objects. Objects interact using method calls. So how do we design these methods?

We will look at

- Program design, modular programming, pseudocode
- A design example, asterisk rectangles
- More examples of methods.

Introduction

You now know both the syntax and semantics of a useful subset of Java.

Programmers from the 60's and 70's had a problem designing code.

Programs that were longer than 100 lines of code (loc) were difficult to write, programs of 1000 lines were impossible.

Then came modular programming. Now 10,000 lines is easy, 100,000 is possible.

Program Design: steps to success

- 1. Identify and understand the problem.
- 2. Identify the input and output. Make it concrete (think of an example).
- 3. Solve it manually first.
- 4. Use modular programming.
- 5. Test it. Make sure it works.

Modular Programming

- Think of the overall problem, don't get caught up in the details (Top-Down design).
- Break a problem into modules (subproblems).
- How? Use pseudocode.

Pseudocode

- Pseudocode is a cross between English and a programming language.
- Use pseudocode to design a solution to the problem.
- The pseudocode should be between 2 and 8 lines long.
- Don't get bogged down with details.

Modules (Subproblems)

Each line of pseudocode is in turn considered a subproblem, which will be handled the same way, i.e. expanded in a small number of lines of pseudocode.

Eventually we get to the level of Java programming statements.

The number of times you have to do this depends on the size of the problem.

pseudocode

For example: the pseudocode to do the shopping is

```
while(more items on list)
   find next item
   put it in trolley
   cross it off list
```

This in turn could be expanded into more pseudocode

pseudocode is easy

The idea is to capture the essence of the problem without worrying about Java syntax

pseudocode example

- Problem: print a Rectangle of asterisks
- Write a program that takes two numbers (*r* the number of rows and *c* the number of columns), and prints a rectangle of asterisks *r* rows by *c* columns. E.g.

```
Enter the rows and columns: 2 4
****
****
```

First Steps:

- Read and understand the problem
- Make it concrete. Try actual values of r and c. E.g. let's try r = 2 and c = 7
- Solve it manually. In this case the program would print

```
*******
*******
```

Next: Modular programming

- Break the problem into manageable chunks. For example, one chunk could be to print a row of asterisks. Use pseudocode to help

```
print 7 asterisks
print 7 asterisks
```

- This looks like it might do the job when there are 2 rows and 7 columns. In fact there is a bug in this.

You need to print a new line after each row

In the general case

- To handle a different number of rows and columns we need a for loop.

```
for i = 1 to r
   print c asterisks
   print newline
```

- You'll also need to get *r* and *c* from the user.

```
get r and c from user
for i = 1 to r
   print c asterisks
   print newline
```

Translate pseudocode to Java

- The first line is easy:

```
get r and c from user
```

becomes

```
System.out.println("Enter r and c:");
r = Console.readInt();
c = Console.readInt();
```

Translate pseudocode to Java

The following is a little trickier:

```
print c asterisks
print new line
```

To print c asterisks you can use a loop to print one asterisk *c* times. A for loop is used when you know how many times the loop is executed.

```
for i = 1 to c
   print one asterisk
```

Translate pseudocode to Java

This can neatly be implemented with a method

```
void printRowOfAsterisks(int c)
{
   for(int i = 0; i < c; i++)
      System.out.print('*');
}
```

Now the program is simply

```
public class Rectangle
{
   public static void main(String [] a)
   {
      System.out.println("Enter r and c:");
      r = Console.readInt();
      c = Console.readInt();
      for(int i = 0; i < r; i++)
      {
         printRowOfAsterisks(c);
         System.out.println();
      }
   }
}
```

Translate pseudocode to Java

- Note that the program is now quite readable. This is because it is a direct translation of the pseudocode. And the method printRowOfAsterisks() could be used again.
- When you have typed in the program, compile it (fix any compile errors).
- Run the program and test it with various values of *r* and *c*. Try possibly awkward values, e.g. r = 0 and -4.

Design: Jargon

This process of design has many names:

- problem decomposition
- modularisation
- incremental development
- stepwise refinement
- abstraction

A little more on methods

More method examples: Maximum

```
int max(int a, int b)
{
   if(a > b)
      return a; // a larger => return a;
   else
      return b;
}
```

```
public class Teens
{
   public static boolean isTeenager(int age)
   {
      if(age >= 13 && age <= 19)
         return TRUE;
      else
         return FALSE;
   }

   public static void main(String [] args)
   {
      println("Enter your age:");
      int yourAge = Console.readInt();

      if(isTeenager(yourAge))
         println("You are a teenager");
      else
         println("You're not a teenager");
   }
}
```

Notes: Teenager

The method isTeenAger() could have been written more concisely as

```
public static boolean isTeenager(int age)
{
   return age >= 13 && age <= 19;
}
```

boolean expression

Methods: General Comments

- Methods should perform one task well. The method's name should reflect that task.
- If a method is too long or complicated, it should be decomposed into smaller methods.
- Methods should be general.
- Methods aid debugging. By testing each method in turn you can quickly locate a problem.