```java
class Lecture3 {
    "Selections"
}

// Keywords
if, else, else if, switch, case, default
```
The basic algorithm (and program) is constituted by the following operations:

- **Sequential statements**: execute instructions in order.
- **Selection**: first check if the predetermined condition is satisfied, then execute the corresponding instruction.
- **Repetition**: repeat the execution of some instructions until the criterion fails.
• Note that they are involved with each other generally.
• For example, recall how to find the maximum in the input list?
Selections

- One-way \texttt{if} statements
- Two-way \texttt{if-else} statements
- Nested \texttt{if} statements
- Multiway \texttt{if-else if-else} statements
- \texttt{switch-case} statements
- Conditional operators
A one-way if statement executes an action if and only if the condition is true.
• The keyword if is followed by the parenthesized condition.
• The condition should be a boolean expression or a boolean value.
• If the condition is true, then the statements in the selection body will be executed once.
• If not, then the program won’t enter the selection body and skip the whole selection body.
• Note that the braces can be omitted if the block contains only single statement.
Example

Write a program which receives a nonnegative number as input for the radius of a circle, and determines the area of the circle.

```java
... 
  double area;
  if (r > 0) {
    area = r * r * 3.14;
    System.out.println(area);
  }
... 
```

- However, the world is not well-defined.
A two-way if-else statement decides which statements to execute based on whether the condition is true or false.
true

boolean-expression

false

Statement(s) for the true case

Statement(s) for the false case
Example

Write a program which receives a number as input for the radius of a circle. If the number is nonnegative, then determine the area of the circle; otherwise, output “Not a circle.”

```java
... double area;
if (r > 0) {
    area = r * r * 3.14;
    System.out.println(area);
} else {
    System.out.println("Not a circle.");
}
input.close();
...
Nested if Statements

- For example,

```java
... if (score &ge; 90) System.out.println("A");
    else {
        if (score &ge; 80) System.out.println("B");
            else {
                if (score &ge; 70) System.out.println("C");
                        else {
                            if (score &ge; 60) System.out.println("D");
                                    else System.out.println("F");
                            }
                    }
            }
    }
...`
```
Multi-Way if-else

- Let’s redo the previous problem.

```java
... if (score >= 90) 
    System.out.println("A");
else if (score >= 80)
    System.out.println("B");
else if (score >= 70)
    System.out.println("C");
else if (score >= 60)
    System.out.println("D");
else
    System.out.println("F");
...
```

- An if-elseif-else statement is a preferred format for multiple alternatives, in order to avoid deep indentation and make the program easy to read.
• The order of conditions may be relevant. (Why?)

```java
... if ((score >= 90) && (score <= 100))
    else if ((score >= 80) && (score < 90))
        ...
    else
        ...
```

• The performance may degrade due to the order of conditions. (Why?)
Common Errors

```java
... double area;
if (r > 0);
    area = r * r * 3.14;
System.out.println(area);
...```

Zheng-Liang Lu Java Programming 102 / 137
Example

Generating random numbers

Write a program which generates 2 random integers and asks the user to answer the math expression.

• For example, the program shows $2 + 5 = ?$
• If the user answers 7, then the program reports “Correct.” and terminates.
• Otherwise, the program reports “Wrong answer. The correct answer is 7.” for this case.
• You may use Math.random() for a random value between 0.0 and 1.0, excluding themselves.
Can you extend this program for all arithmetic expressions (i.e., $+, -, \times, \div$)?
Exercise

Find Max

Write a program which determines the maximum value in 3 random integers whose range from 0 to 99.

• How many variables do we need?
• How to compare?
• How to keep the maximum value?
In this case, a scalar variable is not convenient. (Why?)

So we need arrays and loops.
A switch-case structure takes actions depending on the target variable.

```java
... switch (target) {
    case v1:
        // statements
        break;
    case v2:
        .
        .
        case vk:
            // statements
            break;
    default:
        // statements
}
...
- A **switch-case** statement is more convenient than an **if** statement for multiple **discrete** conditions.
- The variable **target**, always enclosed in parentheses, must yield a value of **char**, **byte**, **short**, **int**, or **String** type.
- The value $v_1, \ldots, v_k$ must have the same data type as the variable **target**.
- In each case, a **break** statement is a must.\(^1\)
  - **break** is used to break a construct!
- The **default** case, which is optional, can be used to perform actions when none of the specified cases matches **target**.
  - Counterpart to **else** statements.

\(^1\)If not, there will be a fall-through behavior.
... // RED: 0
// YELLOW: 1
// GREEN: 2
int trafficLight = (int) (Math.random() * 3);
switch (trafficLight) {
  case 0:
    System.out.println("Stop!!!");
    break;
  case 1:
    System.out.println("Slow down!!");
    break;
  case 2:
    System.out.println("Go!!!");
}
...
A conditional expression evaluates an expression based on the specified condition and returns a value accordingly.

```java
... someVar = booleanExpr ? exprA : exprB;
...)
```

- This is the only ternary operator in Java.
- If the `boolean` expression is evaluated `true`, then return `expr A`; otherwise, `expr B`.
For example,

```java
... if (num1 > num2)
    max = num1;
else
    max = num2;
...
```

Alternatively, one can use a conditional expression like this:

```java
... max = (num1 > num2) ? num1 : num2;
...```
class Lecture4 {

    "Loops"

}

// keywords:
while, do, for, break, continue
Loops

A loop can be used to make a program execute statements repeatedly without having to code the same statements.

• For example, a program outputs “Hello, Java.” for 100 times.

```java
... System.out.println("Hello, Java.");
System.out.println("Hello, Java.");
.
  // copy and paste for 100 times
  System.out.println("Hello, Java.");
...
```
This is a simple example to show the power of loops.

In practice, any routine which repeats couples of times\(^2\) can be done by folding them into a loop.

\(^2\)I prefer to call them “patterns.”
Loops provide substantial computational power.
Loops bring an efficient way of programming.
Loops could consume a lot of time.\(^3\)

\(^3\)We will visit the analysis of algorithms in the end of this lecture.
while Loops

A while loop executes statements repeatedly while the condition is true.

```java
... while (condition) {
    // loop body
}
...
```

- The condition should be a boolean expression which determines whether or not the execution of the body occurs.
- If true, the loop body is executed and check the condition again.
- Otherwise, the entire loop terminates.
Write a program which sums up all integers from 1 to 100.

- In math, the question can be written as:

\[
\text{sum} = 1 + 2 + \cdots + 100.
\]

- But this form is not doable in the machine.\(^4\)

---

\(^4\)We need to develop computational thinking. Read http://rsta.royalsocietypublishing.org/content/366/1881/3717.full or http://blog.orangeapple.tw/posts/what-is-computational-thinking/.
• Normally, the machine executes the instructions **sequentially**.
• So one needs to decompose the math equation into several steps, like:

```java
...  
    int sum = 0;
    sum = sum + 1;
    sum = sum + 2;
    ...
    sum = sum + 100;
...  
```

• It is obvious that many similar statements can be found.
• Using a **while** loop, the program can be rearranged as follows:

```java
... int sum = 0;
int i = 1;
while (i <= 100) {
    sum = sum + i;
    ++i;
}
...```

• You should guarantee that the loop will terminate as expected.
• In practice, the number of loop steps (iterations) is **unknown** until the input data is given.
Malfunctioned Loops

• It is really easy to make an infinite loop.

```
... while (true);
...
```

• The common errors of the loops are:
  • never start
  • never stop
  • not complete
  • exceed the expected number of iterations
Example

Write a program which asks the sum of two random integers and lets the user repeatedly enter a new answer until correct.

```java
... 
Scanner input = new Scanner(System.in);
int x = (int) (Math.random() * 10);
int y = (int) (Math.random() * 10);
int ans = x + y;

System.out.println(x + " + " + y + " = ? ");
int z = input.nextInt();

while (z != ans) {
    System.out.println("Try again? ");
    z = input.nextInt();
}
System.out.println("Correct.");
input.close();
...
```
Loop Design Strategy

- Writing a correct loop is not an easy task for novice programmers.
- Consider 3 steps when writing a loop:
  - Find the pattern: identify the statements that need to be repeated.
  - Wrap by loops: put these statements in the loop.
  - Set the continuation condition: translate the criterion from the real world problem into computational conditions.\(^5\)

\(^5\)Not unique.
Another common technique for controlling a loop is to designate a special value when reading and processing a set of values.

- This special input value, known as a *sentinel value*, signifies the end of the loop.
- For example, the operating systems and the GUI apps.
Example: Cashier Problem

Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```java
... int total = 0;
Scanner input = new Scanner(System.in);

System.out.println("Enter price?");
int price = input.nextInt();
while (price > 0) {
    total += price;
    System.out.println("Enter price?");
    price = input.nextInt();
    // These two lines above repeat Line 3 and 4?!
}

System.out.println("Total = " + total);
input.close();
...
do-while Loops

A do-while loop is similar to a while loop except that it does execute the loop body first and then checks the loop continuation condition.

```java
... do {
    // loop body
} while (condition); // Do not miss the semicolon!
...
```

- Note that there is a semicolon at the end of the do-while loop.
- The do-while loops are also called posttest loops, in contrast to while loops, which are pretest loops.
Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```java
... int total = 0, price = 0;
Scanner input = new Scanner(System.in);

do {
    total += price;
    System.out.println("Enter price?");
    price = input.nextInt();
} while (price > 0);

System.out.println("Total = " + total);
input.close();
...
A for loop generally uses a variable to control how many times the loop body is executed.

```java
... for (init_action; condition; increment) {
    // loop body
}
...
```

- **init-action**: declare and initialize a variable
- **condition**: set a criterion for loop continuation
- **increment**: how the variable changes after each iteration
- Note that these terms are separated by semicolons.
Example

Sum from 1 to 100

Write a program which sums from 1 to 100.

```java
...  
   int sum = 0;
   for (int i = 1; i <= 100; ++i)
       sum = sum + i;
...  
```

• Compared to the while version,

```java
...  
   int sum = 0;
   int i = 1;
   while (i <= 100) {
       sum = sum + i;
       ++i;
   }
...  
```
Flowchart for a loop:

1. Initial-Action
2. **loop-continuation-condition?**
   - true: Statement(s) (loop body)
   - false: action-after-each-iteration

The flowchart represents a loop where a condition is checked at each iteration. If the condition is true, the loop body is executed. If the condition is false, the loop continues from the beginning.
Example: Selection Resided in Loop

Display all even numbers

Write a program which displays all even numbers smaller than 100.

- An even number is an integer of the form \( x = 2k \), where \( k \) is an integer.
You may use the modular operator (%).

```java
... 
for (int i = 1; i <= 100; i++) {
    if (i % 2 == 0) System.out.println(i);
}
...
```

Also consider this alternative:

```java
... 
for (int i = 2; i <= 100; i += 2) {
    System.out.println(i);
}
...
```

How about odd numbers?
Example: Monte Carlo Simulation\textsuperscript{6}

- Write a program which conducts a Monte Carlo simulation to estimate $\pi$.

\textsuperscript{6}See https://en.wikipedia.org/wiki/Monte_Carlo_method.
Jump Statements

The keyword **break** and **continue** are often used in repetition structures to provide additional controls.

- **break**: the loop is **terminated** right after a **break** statement is executed.
- **continue**: the loop **skips** this iteration right after a **continue** statement is executed.
- In practice, jump statements in loops should be conditioned.
Example: Primality

Write a program which determines if the input integer is a prime number.

• Let $x > 1$ be any natural number.
• Then $x$ is said to be a prime number if $x$ has no positive divisors other than 1 and itself.
• It is then straightforward to check if it is prime by dividing $x$ by all natural numbers smaller than $x$.
• For speedup, you can divide $x$ by only numbers smaller than $\sqrt{x}$. (Why?)
Scanner input = new Scanner(System.in);
System.out.println("Enter x > 2?");
int x = input.nextInt();
boolean isPrime = true;
input.close();

double upperBd = Math.sqrt(x);
for (int y = 2; y < upperBd; y++) {
    if (x % y == 0) {
        isPrime = false;
        break;
    }
}

if (isPrime) {
    System.out.println("Prime");
} else {
    System.out.println("Composite");
}