Introduction to Java for C++ Programmers
JAC444

Week 02
Elementary Programming with Selections & Loops

Instructor

Hossein Pourmodheji

hossein.pourmodheji@senecacollege.ca

Identifiers

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).

- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
  - An identifier cannot be a keyword/reserved word.
    - **keywords** are reserved for use by Java and are always spelled with all **lowercase** letters.
  - An identifier **cannot be** true, false, or null.
  - An identifier can be of any length.

Declaring Variables

```
int x;              // Declare x to be an
                    // integer variable;
double radius;      // Declare radius to
                    // be a double variable;
char a;             // Declare a to be a
                    // character variable;
```

Assignment Statements

```
x = 1;              // Assign 1 to x;

radius = 1.0;   // Assign 1.0 to radius;

a = 'A';            // Assign 'A' to a;
```

Declaration and Assignment in One Step

```
int x = 1;

double d = 1.4;
```

Constants

```
final datatype CONSTANTNAME = VALUE;


final double PI = 3.14159;
final int SIZE = 3;
```

Naming Conventions

- Choose meaningful and descriptive names.

- **Variable** and **method** names use lowercase.
  - If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables radius and area, and the method computeArea.

- Constants:
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant PI and MAX_VALUE

Data Types

Type	Size	Range	Default
boolean	1 bit	true or false	false
byte	8 bits	[-128, 127]	0
short	16 bits	[-32,768, 32,767]	0
char	16 bits	['\u0000', '\uffff'] or [0, 65535]	'\u0000'
int	32 bits	[-2,147,483,648 to 2,147,483,647]	0
long	64 bits	$[-2^{63}, 2^{63}-1]$	0
float	32 bits	32-bit IEEE 754 floating-point	0.0
double	64 bits	64-bit IEEE 754 floating-point	0.0

Numeric Operators

Name	*Meaning*	*Example*	*Result*
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Shortcut Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

Integer Division

+, -, *, /, and %

5 / 2 yields an integer 2

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)

The String Type

- The char type only represents one character. To represent a string of characters, use the data type called String. For example,

```
String message = "Welcome to Java";
```

- String is actually a predefined class in the Java. The String type is not a primitive type.

String Concatenation

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes
SupplementB
```

Relational Operators

Java Operator	*Mathematics Symbol*	*Name*	*Example (radius is 5)*	*Result*
`<`	<	less than	`radius < 0`	`false`
`<=`	≤	less than or equal to	`radius <= 0`	`false`
`>`	>	greater than	`radius > 0`	`true`
`>=`	≥	greater than or equal to	`radius >= 0`	`true`
`==`	=	equal to	`radius == 0`	`false`
`!=`	≠	not equal to	`radius != 0`	`true`

radius = 5

Logical Operators

Operator	*Name*	*Description*
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction

Truth Table for Operator !

p	!p	*Example (assume* `age = 24, weight = 140`*)*
`true`	`false`	`!(age > 18)` is `false`, because `(age > 18)` is `true`.
`false`	`true`	`!(weight == 150)` is `true`, because `(weight == 150)` is `false`.

Truth Table for Operator &&

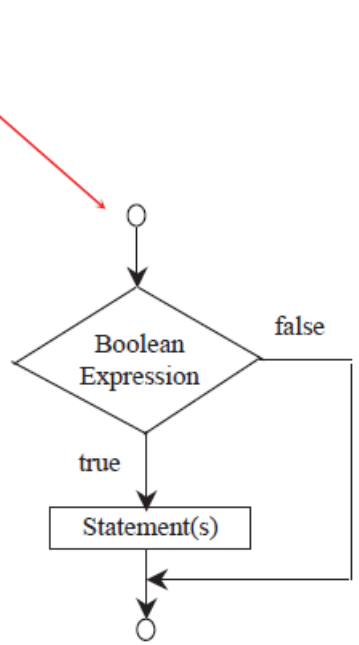
p_1	p_2	p_1 && p_2	*Example (assume* `age = 24, weight = 140`*)*
`false`	`false`	`false`	
`false`	`true`	`false`	`(age > 28) && (weight <= 140)` is `true`, because `(age > 28)` is `false`.
`true`	`false`	`false`	
`true`	`true`	`true`	`(age > 18) && (weight >= 140)` is `true`, because `(age > 18)` and `(weight >= 140)` are both `true`.

Truth Table for Operator ||

p_1	p_2	p_1 || p_2	*Example (assume* `age = 24, weight = 140`*)*
`false`	`false`	`false`	`(age > 34) || (weight >= 150)` is `false`, because `(age > 34)` and `(weight >= 150)` are both `false`.
`false`	`true`	`true`	
`true`	`false`	`true`	`(age > 18) || (weight < 140)` is `true`, because `(age > 18)` is `true`.
`true`	`true`	`true`	

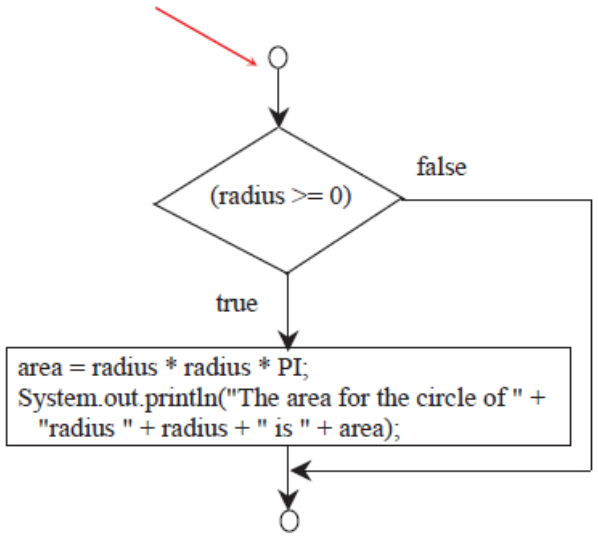
One-way `if` Statements

```
if (boolean-expression) {
  statement(s);
}
```



(A)

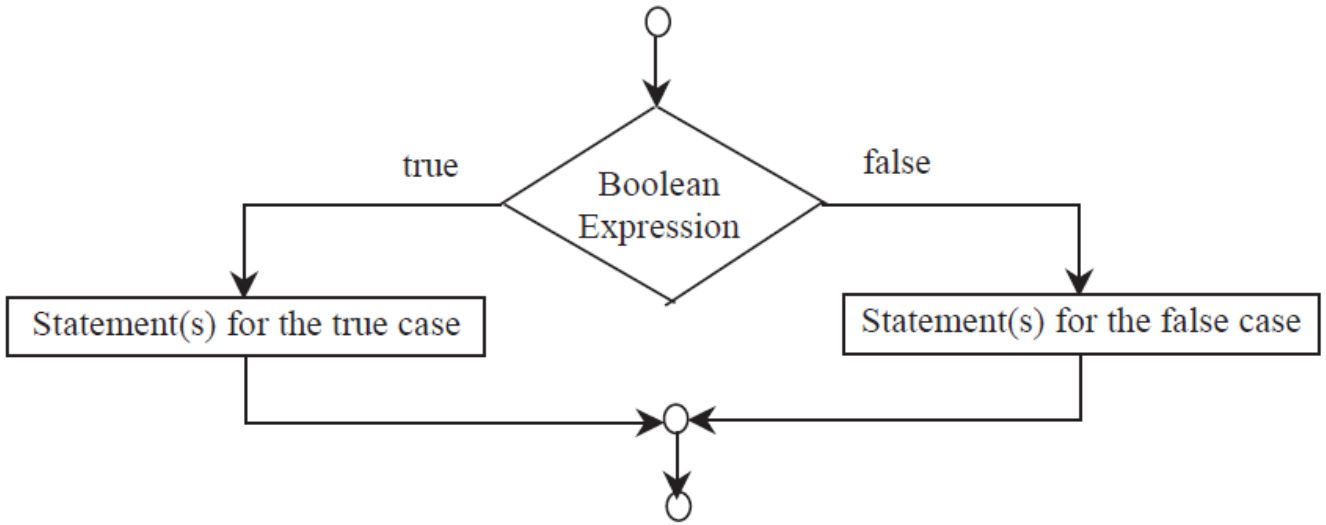
```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area"
    + " for the circle of radius "
    + radius + " is " + area);
}
```



(B)

The Two-way `if` Statement

```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```



if...else Example

```
if (radius > 0) {
    area = radius * radius * 3.14159;

    System.out.println("The area for the "
      + "circle of radius " + radius +
      " is " + area);
}
else {
    System.out.println("Non-positive input");
}
```

Note

```
if i > 0 {
  System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(b) Correct

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(a)

Equivalent

```
if (i > 0)
  System.out.println("i is positive");
```

(b)

Multiple Alternative `if` Statements

```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

Equivalent

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Note

- The else clause matches the most recent if clause in the same block.

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

(a)

Equivalent

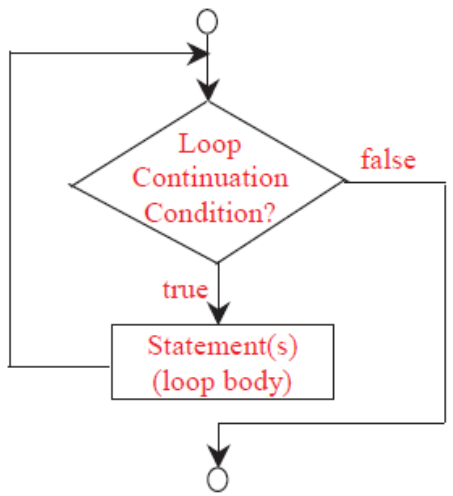
```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

(b)

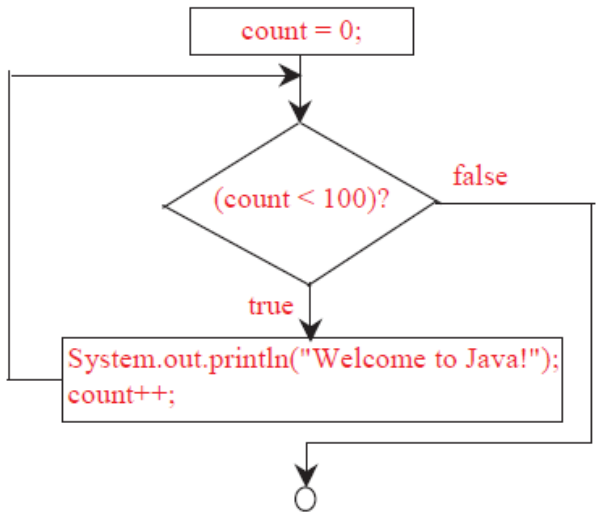
while Loop Flow Chart

```
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```



(A)

```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```



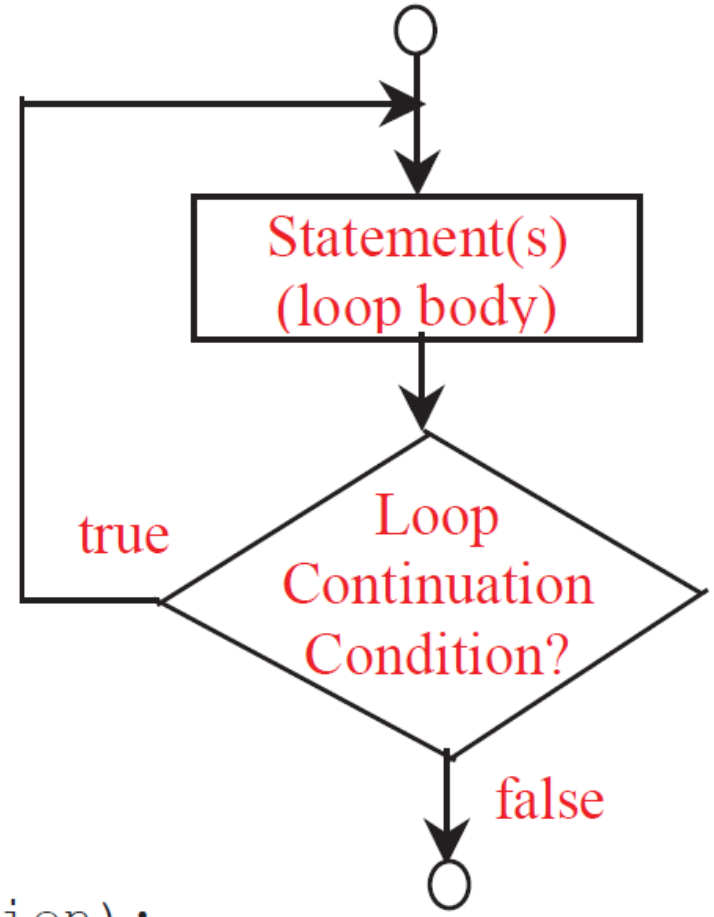
(B)

Introducing while Loops

```
int count = 0;
while (count < 100) {
        System.out.println("Welcome to Java");
        count++;
}
```

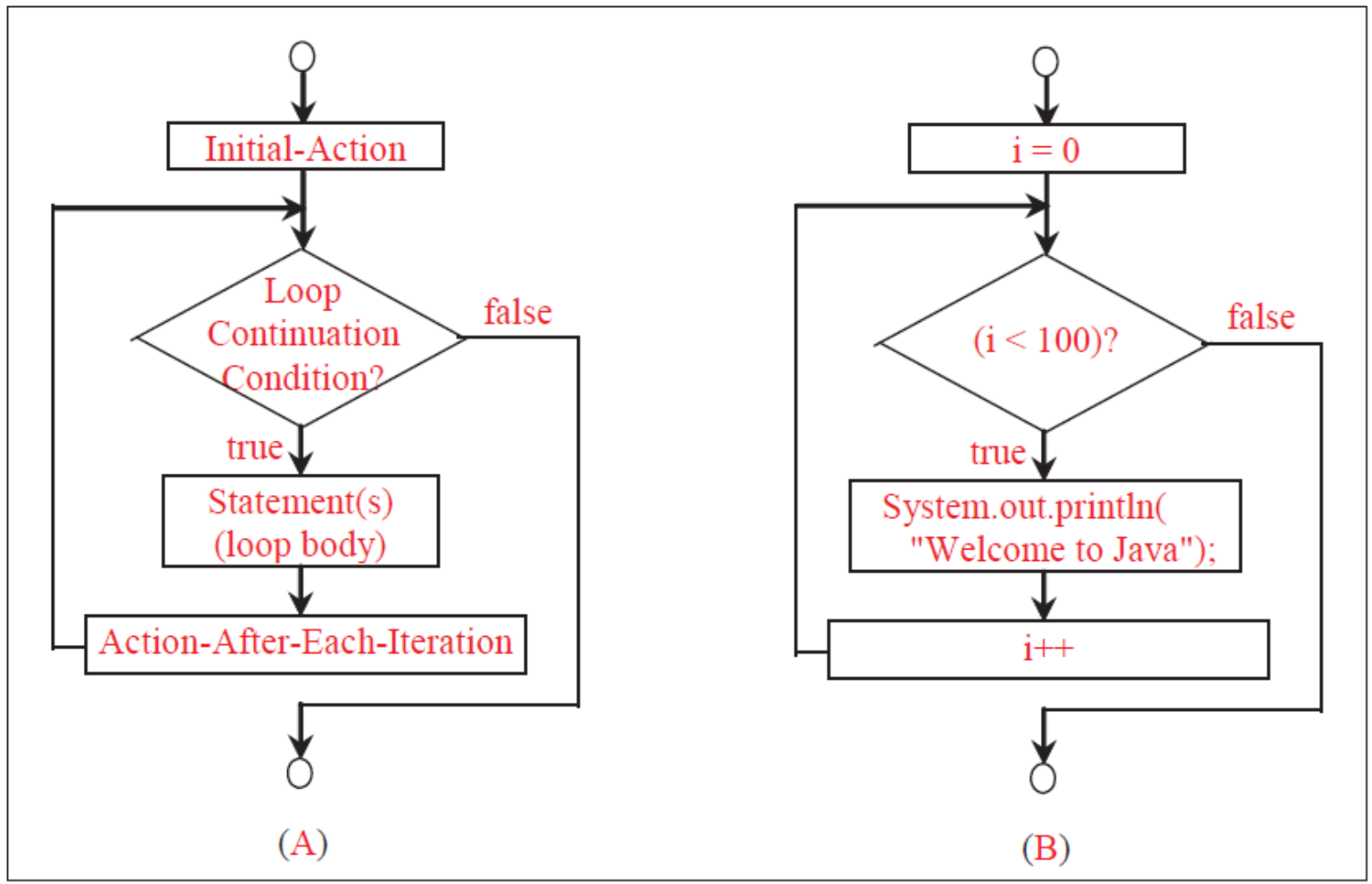
do-while Loop

```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```



```
for (initial-action; loop-
    continuation-condition;
    action-after-each-iteration) {
  // loop body;
  Statement(s);
}
```

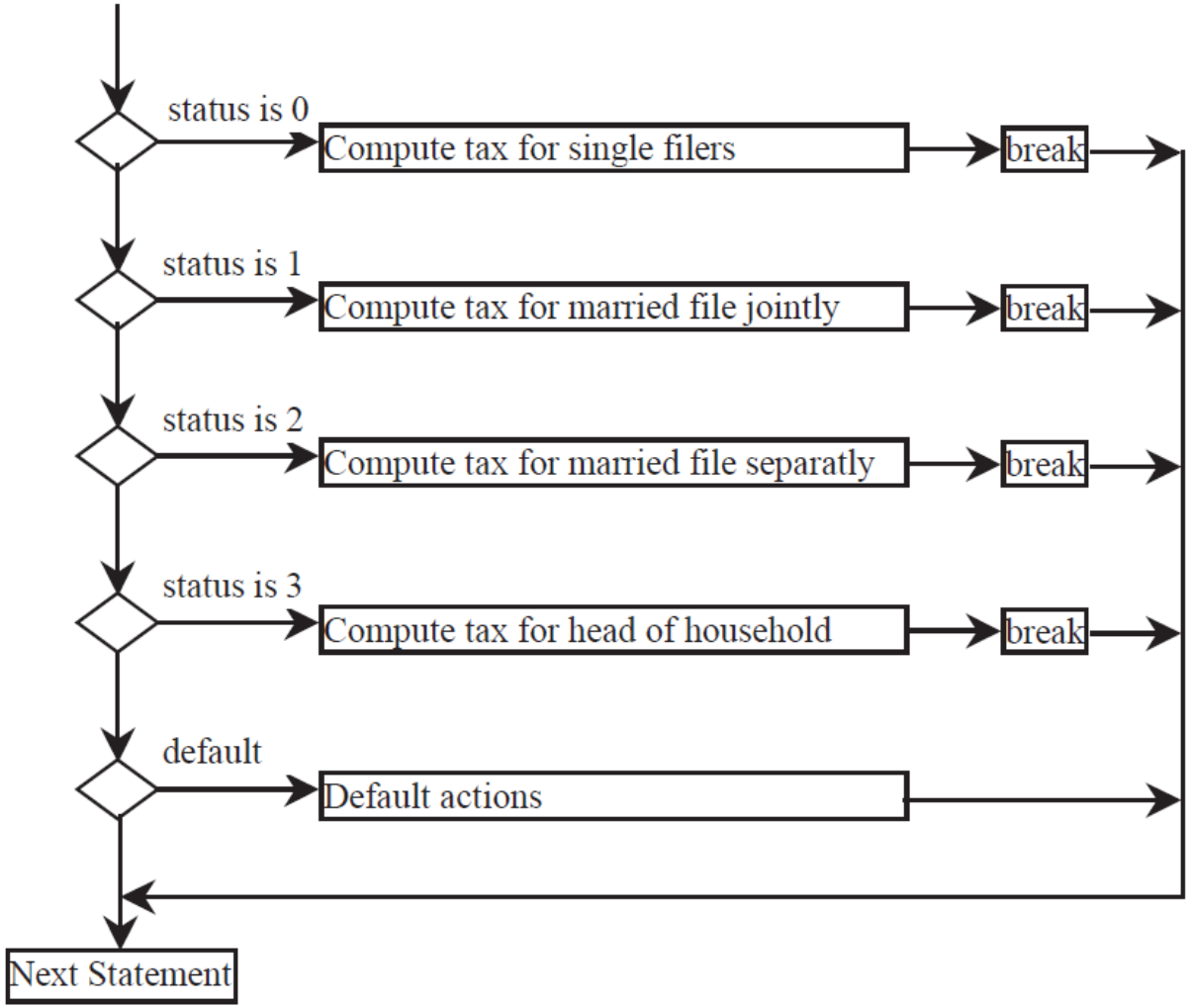
```
int i;
for (i = 0; i < 100; i++) {
  System.out.println(
    "Welcome to Java!");
}
```



switch Statements

```
switch (status) {
   case 0: do something here;
           break;
   case 1: do something here;
           break;
   case 2: do something here;
           break;
   case 3: do something here;
           break;
   default: System.out.println("Errors: invalid status");
           System.exit(0);
}
```

switch Statement Flow Chart



Type Casting

```
Implicit casting
    double d = 3; (type widening)
Explicit casting
    int i = (int)3.0; (type narrowing)
    int i = (int)3.9; (Fraction part is truncated)
```

What is wrong? `int x = 5 / 2.0;`