Statements

- Variable declaration

```
type identifier;
```

- Assignment

```
variable = expression;
```

- User Interface: output

```
System.out.println(string_expression);
```

- User Interface: input

```
variable = scanner.nextType();
```

Primitive Data Types

General category	Type	Description	Examples
Numeric	int	Integers	0,1,-3
	long	Long integers	65537l
	short	Short integers	2,-6
	byte	Bytes	255
	float	Rationals	1.33f
	double	Rationals	1.618
Text	char	Single characters	’x’, ’ ’
	String	Sequences of characters	“abc”
Logic	boolean	Truth values	true, false

Assignment

- If *variable* is of numeric type (int, float, etc.)

```
variable = arithmetic_expression;
```

- If *variable* is of String type

```
variable = string_expression;
```

- If *variable* is of boolean type

```
variable = boolean_expression;
```

Boolean expressions

```
true

false

true && true

false || true

!false

!true && false || !false

!(sunny && false)

5 < 7

6 >= 8

2 + x == 9 && b < 8 || c == true
```

Syntax of boolean expressions

- Boolean operators:

Operator	Name	Meaning
&&	and	both operands are true
||	or	one of the operands is true
!	not	the opposite of the operand

- Relational operators:

Operator	Name
<	(Strictly) less than
>	(Strictly) greater then
==	Equal to
<=	Less or equal to
>=	Greater or equal to
!=	Not equal to

Precedence

Precedence	Operator	Operation	Associativity
1	+	Unary plus	right to left
	-	Unary minus	
	!	Logical negation (NOT)	
2	*	Multiplication	left to right
	/	Division	
	%	Remainder (modulo)	
3	+	Addition	left to right
	-	Substraction	
	+	String concatenation	
4	<	Less than	Left to right
	<=	Less than or equal to	
	>	Greater than	
	>=	Greater than or equal to	
5	==	Equals to	Left to right
	!=	Different to	
6	&&	Logical conjunction (AND)	Left to right
7	||	Logical disjunction (OR)	Left to right

Semantics of expressions

- The meaning of an expression is the value of the expression
  - An arithmetic expression is evaluated to a number
  - A string expression is evaluated to a string
  - A boolean expression is evaluated to a truth-value (true or false)

Semantics of boolean expressions

- Truth tables
- Assume that a and b are boolean expressions

a	!a
true	false
false	true

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Semantics of boolean expressions

- The value of

```
true && false || !false
```

is the same as the value of

```
(true && false) || (!false)
```

which is

```
(true && false) || true
```

which is

```
false || true
```

which is

```
true
```

Semantics of boolean expressions

- Exclusive or: either a is true or b is true but not both

  a && !b || !a && b

a	b	!b	a&&!b	!a	!a&&b	a&&!b || !a&&b
true	true	false	false	false	false	false
true	false	true	true	false	false	true
false	true	false	false	true	true	true
false	false	true	false	true	false	false

Semantics of boolean expressions

- What is the value of 4+x==9 && b < 8 || !c ?

Semantics of boolean expressions

- What is the value of 4+x==9 && b < 8 || !c ?

- It depends on the values of the relational expressions (4+x==9), (b<8) and the boolean expression c.

- These expressions depend on the values of x, b and c, which we do not know

- ...but we can consider the all the possible truth values for each subexpression:

Semantics of boolean expressions

4+x==9	b<8	c	!c	4+x==9 && b<8	(((4+x)==9)&&(b<8))||(!c)
true	true	true	false	true	true
true	true	false	true	true	true
true	false	true	false	false	false
true	false	false	true	false	true
false	true	true	false	false	false
false	true	false	true	false	true
false	false	true	false	false	false
false	false	false	true	false	true

Semantics of boolean expressions

temp > -20.0 || !windy && sunny

temp>-20.0	windy	sunny	!windy	!windy && sunny	(temp > -20.0) || (!windy && sunny)
true	true	true	false	false	true
true	true	false	false	false	true
true	false	true	true	true	true
true	false	false	true	false	true
false	true	true	false	false	false
false	true	false	false	false	false
false	false	true	true	true	true
false	false	false	true	false	false

Note about variables

- The name of a variable is just a symbolic name to make the program more readable

- The name of the variable does not give the variable any special meaning

```
double temp = -27.0;
boolean cold;
cold = temp <= -20.0;
```

- Does not mean that it is actually cold!

- It only means

```
double x = -27.0;
boolean y;
y = x <= -20.0;
```

- But it is useful for the programmer to give variables meaningful names

Conditionals

- The conditional statement is a statement used to make decisions: take different courses of action depending on some given condition

- There are several (syntactic) forms of conditionals

- The simplest form is:

```
  if (boolean_expression) {
      list_of_statements;
  }
```

- For example:

```
if (winter && temperature >= -30.0f) {
    System.out.print(“I’m going ”);
    System.out.println(“skiing!”);
}
```

Conditionals

- Conditionals are statements, so they go inside methods

```
public class SomeProgram {
  public static void main(String[] args)
  {
    boolean winter = true;
    float temperature = -10.0f;

    if (winter && temperature >= -10.0f) {
      System.out.print(“I’m going ”);
      System.out.println(“skiing!”);
    }
  }
}
```

Conditionals

- The following is very, very, very, very very, very, very wrong!

```
public class SomeProgram {
    boolean winter = true;
    float temperature = -10.0f;

    if (winter && temperature >= -10.0f) {
      System.out.print(“I’m going ”);
      System.out.println(“skiing!”);
    }
}
```

Conditionals

- Semantics

- A conditional

```
if (condition) {
    list_of_statements
}
```

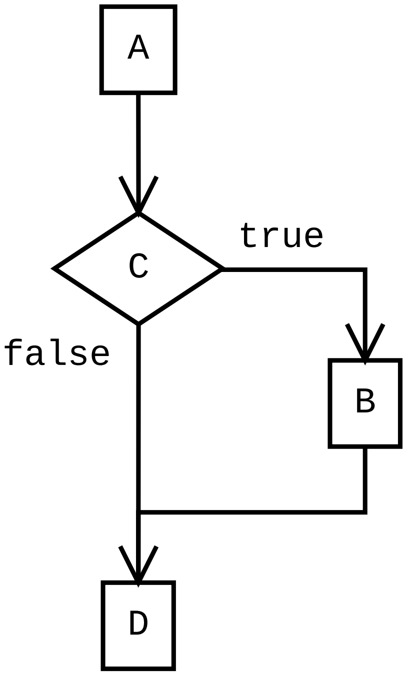
  is executed as follows:

  1. The *condition* is evaluated, yielding `true` or `false`
  2. If the result of evaluating the condition is `true`, then
     (a) the list of statements is executed,
     (b) and when it finnishes, computation continues directly after the conditional
  3. Otherwise, if the result of evaluating the condition is `false`, then
     (a) the list of statements is ignored,
     (b) and computation continues directly after the conditional

Conditionals

```
A;
if (C) {
    B;
}
D;
```

- Control flow diagram



Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
The house is, hmm... safe
The policeman eats donuts
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(‘‘The house is, hmm...’’);
    System.out.println(‘‘ safe’’);
}
System.out.println(‘‘The policeman eats donuts’’);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
The policeman eats donuts
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(‘‘The house is, hmm...’’);
    System.out.println(‘‘ safe’’);
}
System.out.println(‘‘The policeman eats donuts’’);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;
if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

- Conditionals with alternatives

```
if (boolean_expression) {
     list_of_statements_1;
}
else {
     list_of_statements_2;
}
```

- For example:

```
if (!raining) {
     System.out.println(“Go out”);
}
else {
     System.out.println(“Stay in”);
}
```

Conditionals

- A conditional

  ```
  if (condition) {
      list_of_statements_1
  }
  else {
      list_of_statements_2
  }
  ```

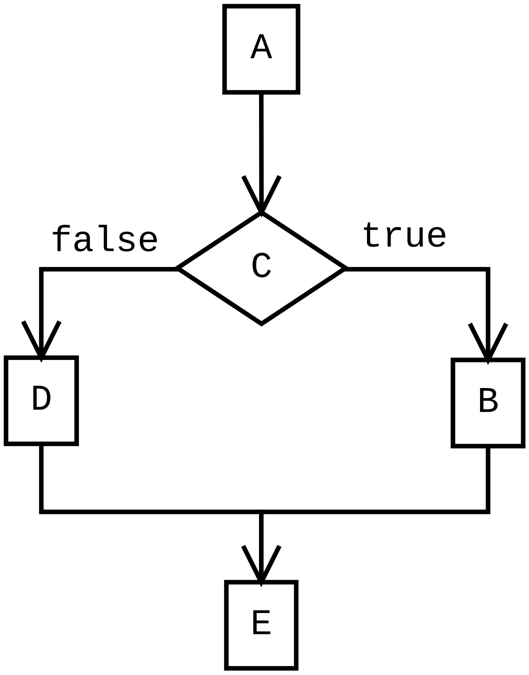
  is executed as follows:

  1. The *condition* is evaluated, yielding `true` or `false`
  2. If the result of evaluating the condition is `true`, then
     (a) the list of statements 1 is executed,
     (b) and when it finnishes, computation continues directly after the conditional
  3. Otherwise, if the result of evaluating the condition is `false`, then
     (a) the list of statements 2 is executed,
     (b) and when it finnishes, computation continues directly after the conditional

Conditionals

```
A;
if (C) {
    B;
}
else {
    D;
}
E;
```

- Control flow diagram



Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
The house is, hmm... safe
The policeman eats donuts
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(‘‘The house is, hmm...’’);
    System.out.println(‘‘ safe’’);
}
else {
    System.out.print(‘‘The house is ’’);
    System.out.println(‘‘vulnerable’’);
}
System.out.println(‘‘The policeman eats donuts’’);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(‘‘The house is, hmm...’’);
    System.out.println(‘‘ safe’’);
}
else {
    System.out.print(‘‘The house is ’’);
    System.out.println(‘‘vulnerable’’);
}
System.out.println(‘‘The policeman eats donuts’’);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = true;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(‘‘The house is, hmm...’’);
    System.out.println(‘‘ safe’’);
}
else {
    System.out.print(‘‘The house is ’’);
    System.out.println(‘‘vulnerable’’);
}
System.out.println(‘‘The policeman eats donuts’’);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
The house is vulnerable
The policeman eats donuts
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

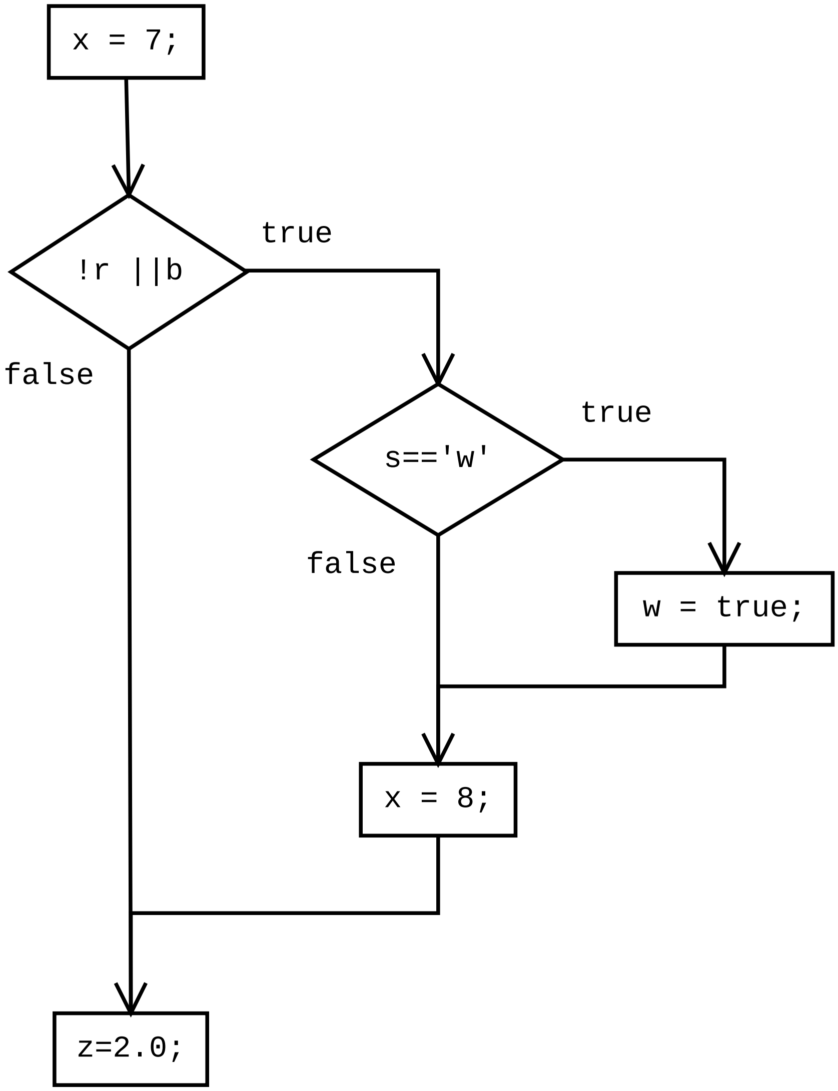
```
boolean alarm_on, lights_on, vicious_dog_is_awake;
alarm_on = false;
lights_on = false;
vicious_dog_is_awake = true;

if (alarm_on || lights_on && vicious_dog_is_awake)
{
    System.out.print(“The house is, hmm...”);
    System.out.println(“ safe”);
}
else {
    System.out.print(“The house is ”);
    System.out.println(“vulnerable”);
}
System.out.println(“The policeman eats donuts”);
```

Conditionals

- Nested conditionals: Conditionals are statements and therefore they can go inside other conditionals

```
x = 7;
if (!r || b) {
    if (s ==‘w’) {
        w = true;
    }
    x = 8;
}
z = 2.0;
```

Conditionals

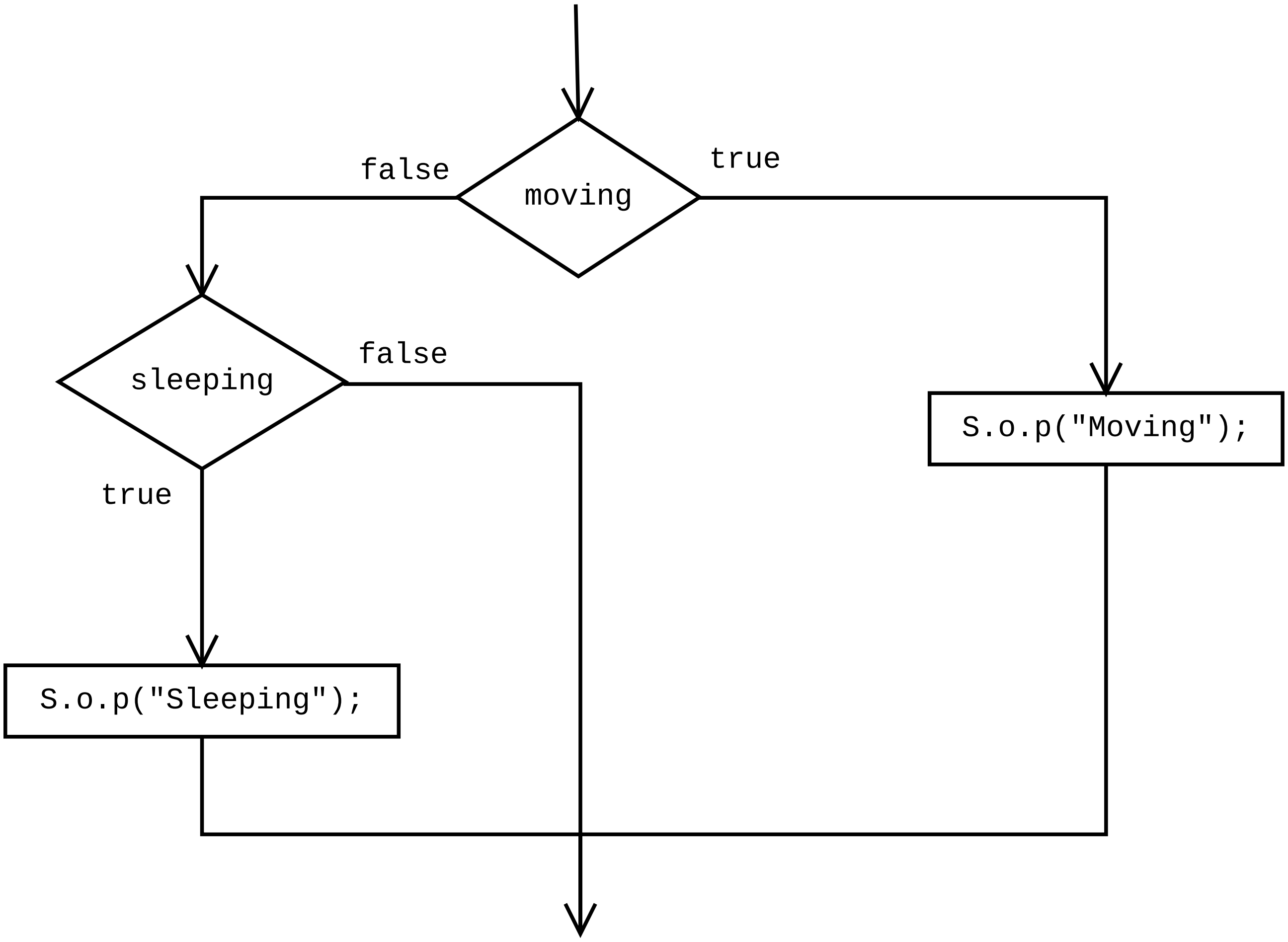


Conditionals

- Nested conditionals

```
if (moving) {
    System.out.println(“Moving”);
}
else {
    if (sleeping) {
        System.out.println(“Sleeping”);
    }
}
```

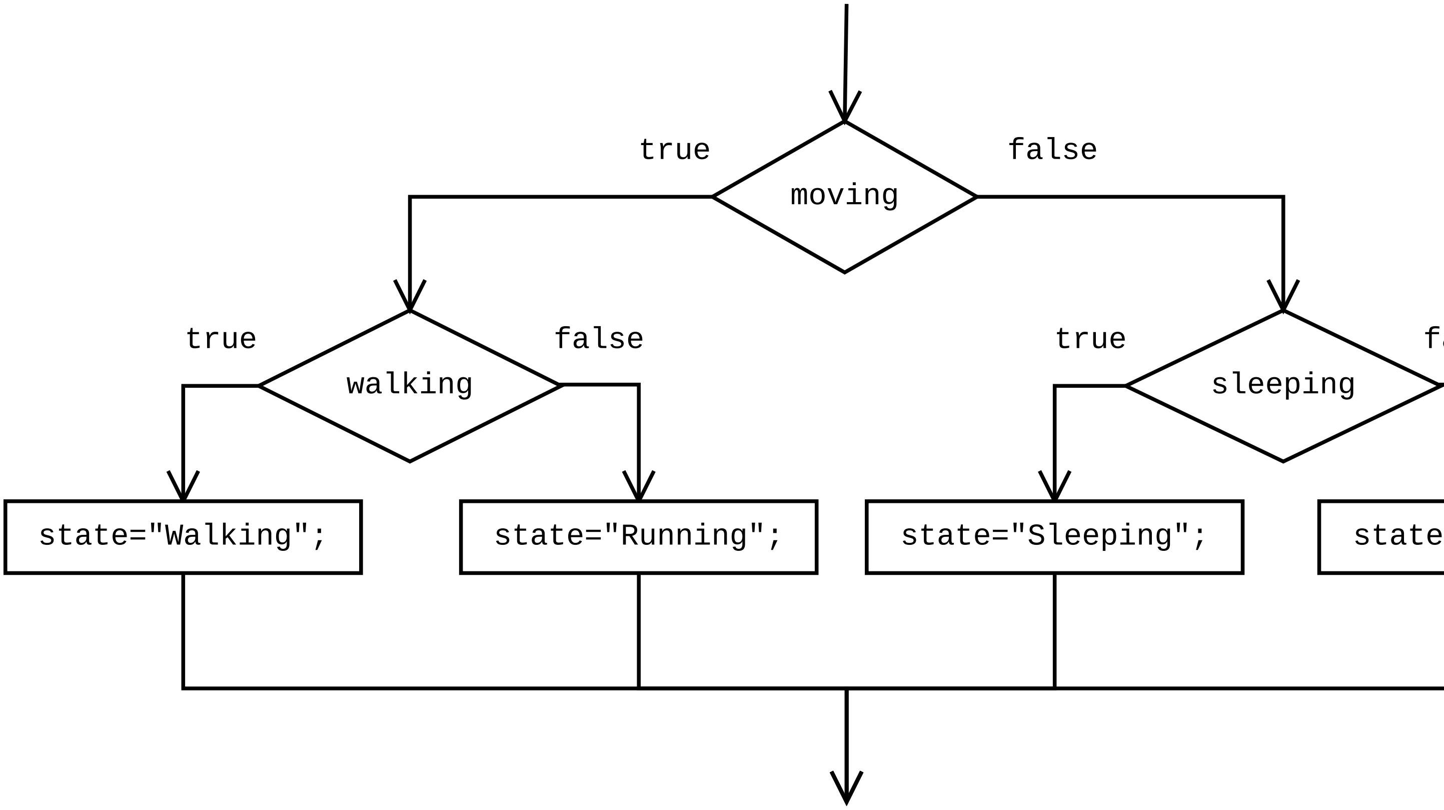
Conditionals



Conditionals

```
if (moving) {
    if (walking) {
        state = “Walking”;
    }
    else {
        state = “Running”;
    }
}
else {
    if (sleeping) {
        state = “Sleeping”;
    }
    else {
        state = “Thinking”;
    }
}
```

Decision tree



Conditionals

```
if (moving) {
    body_active = true;
    if (walking) {
        state = ‘‘Walking’’;
    }
    else {
        state = ‘‘Running’’;
    }
    System.out.println(‘‘Moving’’);
}
else {
    body_active = false;
    if (sleeping) {
        state = ‘‘Sleeping’’;
    }
    else {
        state = ‘‘Thinking’’;
    }
}
```

Conditionals

- Assignment is not equality, and therefore it cannot be used in the condition of a conditional or in any other boolean expression

- An incorrect usage of assignment as equality

```
int x = 4, y = 3;
boolean z = false;
y = y + 1;
if (x = y) { // WRONG
    z = true;
}
```

- A correct use of equality

```
int x = 4, y = 3;
boolean z = false;
y = y + 1;
if (x == y) { // correct
    z = true;
}
```

Conditionals

- Order matters

```
int a;
String b = ‘“’’;
a = 7;
if (a < 10) {
    b = ‘‘first case’’;
    a = a + 4;
}
else {
    b = ‘‘second case’’;
}
// a == 11 and b is ‘‘first case’’
```

Conditionals

- Order matters

```
int a;
String b = ‘“”’;
a = 7;
if (a < 10) {
    b = ‘‘first case’’;
    a = a + 4;
}
if (a == 11) {
    b = ‘‘second case’’;
}
// a == 11 and b is ‘‘second case’’
```

Conditionals

- Order matters

```
int a;
String b = ‘‘’’;
a = 11;
if (a < 10) {
    b = ‘‘first case’’;
    a = a + 4;
}
if (a == 11) {
    b = ‘‘second case’’;
}
// a == 11 and b is ‘‘second case’’
```

Conditionals

- Order matters

```
int a;
String b = ‘‘’’;
a = 12;
if (a < 10) {
    b = ‘‘first case’’;
    a = a + 4;
}
if (a == 11) {
    b = ‘‘second case’’;
}
// a == 12 and b is ‘‘’’
```

Conditionals

- A conditional can take the form

```
if (condition)
    statement;
```

without braces, but only when a unique statement depends on the condition

```
if (a == 2)
  b = 4.0;
  c = true;
```

is the same as

```
if (a == 2) {
  b = 4.0;
}
c = true;
```

and not the same as

```
if (a == 2) {
  b = 4.0;
  c = true;
}
```

Conditionals

- The “dangling else” problem:

```
if (sunny)
    if (warm)
        System.out.print(“go out”);
else
    System.out.print(“stay in”);
```

If `sunny` is `false`, it will not print anything!

Conditionals

- It actually means:

```
if (sunny) {
    if (warm) {
        System.out.print(‘‘go out’’);
    }
    else {
        System.out.print(‘‘stay in’’);
    }
}
```

danglelse1.eps not found!

Conditionals

and not

```
if (sunny) {
    if (warm) {
        System.out.print(“go out”);
    }
}
else {
    System.out.print(“stay in”);
}
```

danglelse2.eps not found!

Conditionals

- Order matters: if the value of a variable changes, then it will only affect conditionals which appear after the assignment

```
int a = 2;
String s = ‘‘hello’’;

if (a == 3) {
    s = ‘‘bye’’;
}
a = 3;
```

Conditionals

- Order matters: if the value of a variable changes, then it will only affect conditionals which appear after the assignment

```
int a = 2;
String s = “hello”;

if (a == 2) {
    s = “bye”;
    a = 5;
}
else {
    s = “again”;
}
```

Conditionals

- Order matters: if the value of a variable changes, then it will only affect conditionals which appear after the assignment

```
int a = 2;
String s = ‘‘start’’;

if (a == 1) {
    s = ‘‘ready’’;
}
if (a == 2) {
    s = ‘‘set’’;
}
if (a == 3) {
    s = ‘‘go’’;
}
a = 3;
```

Conditionals

- Order matters

```
boolean sunny = true, snow = false;
float temperature = -20.0f, windchill = -25.0f;

if (!sunny && temperature > -10.0f && !snow) {
    snow = true;
}
if (!snow && temperature - windchill < 10.0f) {
    sunny = false;
}
if (sunny && snow) {
    sunny = snow;
}
```

Conditionals

```
boolean x;

if (false) {
    x = true;
}
```

Conditionals

```
boolean x;

x = false;
if (false) {
    x = true;
}
```

Conditionals

```
boolean x;

x = true;
if (false) {
    x = true;
}
```

Conditionals

```
boolean x;

x = false;
if (x == false) {
    x = true;
}
```

Conditionals

```
boolean x;

x = false;
if (!x) {
    x = true;
}
```

Conditionals

- A few properties of conditionals: (C is any boolean expression, P, Q, and R are any list of statements.)

  ```
  P;
  if (false) {
      Q;
  }
  R;
  ```

  is equivalent to

  ```
  P;
  R;
  ```

Conditionals

- A few properties of conditionals: (C is any boolean expression, P, Q, and R are any list of statements.)

  ```
  P;
  if (C) {
      Q;
  }
  R;
  ```

  is equivalent to

  ```
  P;
  R;
  ```

  if the value of C is always false

Conditionals

- A few properties of conditionals:

  ```
  System.out.println(“A”);
  if (x == x + 1) {
      System.out.println(“B”);
  }
  System.out.println(“C”);
  ```

  is equivalent to

  ```
  System.out.println("A");
  System.out.println("C");
  ```

  because `x==x+1` is always `false`

Conditionals

- A few properties of conditionals: (`C` is any boolean expression, `P`, `Q`, and `R` are any list of statements.)

  ```
  P;
  if (C) {
      Q;
  }
  R;
  ```

  is equivalent to

  ```
  P;
  Q;
  R;
  ```

  if the value of `C` is always `true`

Conditionals

- A few properties of conditionals:

```
System.out.println(“A”);
if (x + 1 == x + 1) {
    System.out.println(“B”);
}
System.out.println(“C”);
```

is equivalent to

```
System.out.println("A");
System.out.println("B");
System.out.println("C");
```

because `x+1==x+1` is always `true`

Conditionals

- A few properties of conditionals: (C is any boolean expression, P, Q, and R are any list of statements.)

  ```
  P;
  if (C == true) {
      Q;
  }
  R;
  ```

  is equivalent to

  ```
  P;
  if (C) {
      Q;
  }
  R;
  ```

Conditionals

- A few properties of conditionals: (C is any boolean expression, P, Q, and R are any list of statements.)

  ```
  P;
  if (C == false) {
      Q;
  }
  R;
  ```

  is equivalent to

  ```
  P;
  if (!C) {
      Q;
  }
  R;
  ```

Decision trees

- Decision trees as a design technique

- Problem: Given three numbers, determine which one is the smallest

- Analysis:
  - Input: three numbers a, b and c
  - Output: a number m, which is the smallest among a, b and c
  - Definitions:
    - ∗ A number m is the *smallest* among three numbers a, b and c if m is one of the three numbers (i.e. $m = a$, $m = b$, or $m = c$), and it satisfies the condition that it is less or equal than the three numbers (i.e. $m \leqslant a$, $m \leqslant b$, and $m \leqslant c$) Note that these are not strict inequalities.
  - Open issues: what kind of numbers?

Analysis

- If there are open issues we can make assumptions as long as:
  - they are consistent with all aspects of the problem,
  - they make sense, and
  - they do not impose restrictions which modify the problem. (For instance in this case we should not assume that the numbers are all different.)

- Assumptions:
  - Numbers can always be compared (not true if we are not dealing with numbers)

- It is often useful to state the obvious
  - In this example, it is "obvious" that m must be one of the three given numbers, but this is crucial, because we could have a very easy solution: to return a number smaller than all of them. The problem is that this would not be solving the original problem.

Design

- First alternative: consider all possibilities:

1. If $a \leqslant b$ and $b \leqslant c$ then let m be a
2. If $a \leqslant c$ and $c \leqslant b$ then let m be a
3. If $b \leqslant a$ and $a \leqslant c$ then let m be b
4. If $b \leqslant c$ and $c \leqslant a$ then let m be b
5. If $c \leqslant a$ and $a \leqslant b$ then let m be c
6. If $c \leqslant b$ and $b \leqslant a$ then let m be c

- This solution is correct. It covers all possibilities, but it requires 12 comparisons in the worst case. It is not a very smart solution, and it does not scale well.

Implementation

```
import cs1.Keyboard;
public class SmallestFinder {
  public static void main(String[] args)
  {
    double a, b, c, m;

    System.out.print(“Enter the first number:”);
    a = Keyboard.readDouble();
    System.out.print(“Enter the second number:”);
    b = Keyboard.readDouble();
    System.out.print(“Enter the third number:”);
    c = Keyboard.readDouble();

    // Continues below ...
```

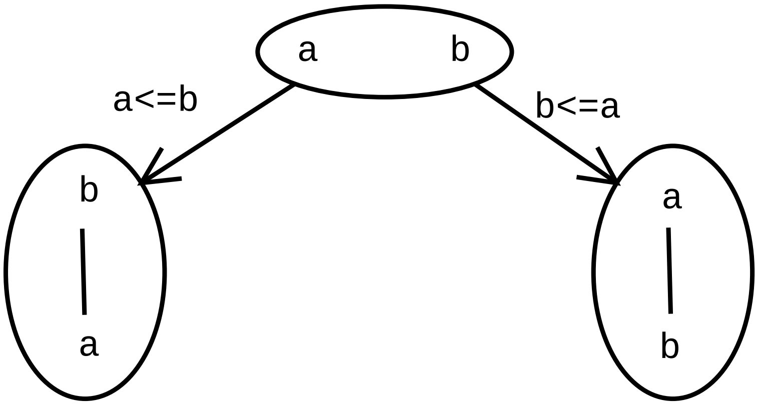
Implementation

```
    if (a <= b && b <= c) m = a;
    if (a <= c && c <= b) m = a;
    if (b <= a && a <= c) m = b;
    if (b <= c && c <= a) m = b;
    if (c <= a && a <= b) m = c;
    if (c <= b && b <= a) m = c;

    System.out.println(‘‘The smallest is ’’ + m);

  }  // End of main method
} // End of SmallestFinder class
```

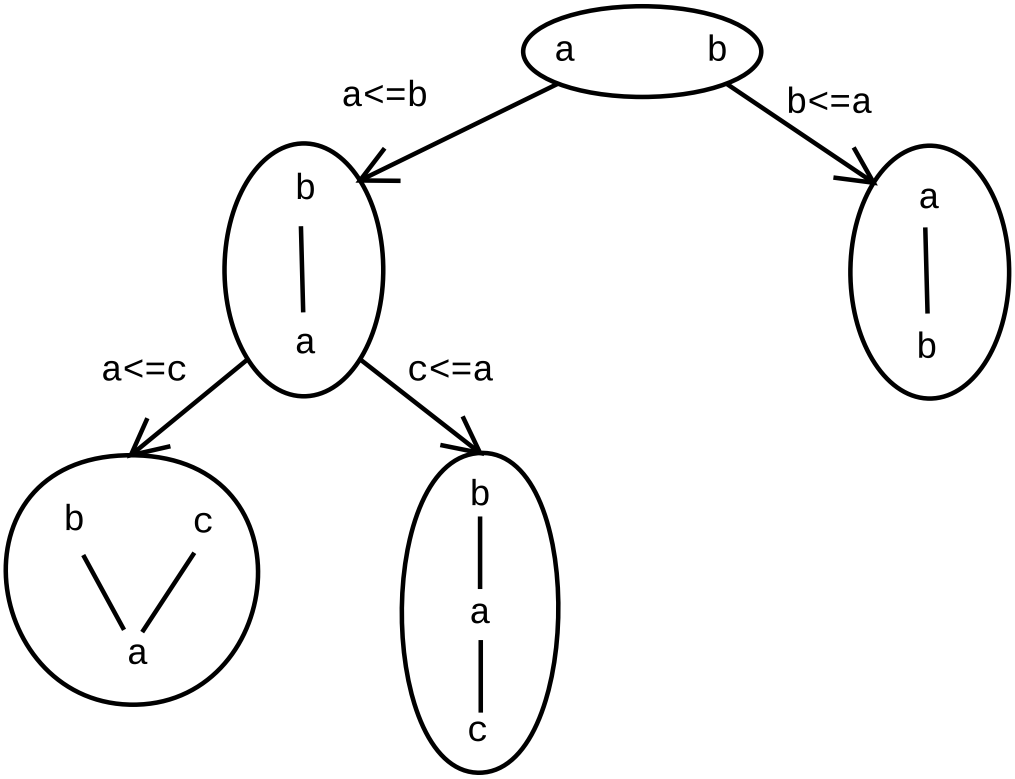
Design

- Second alternative: use decision trees to rule out possibilities, and take into account what is already known.

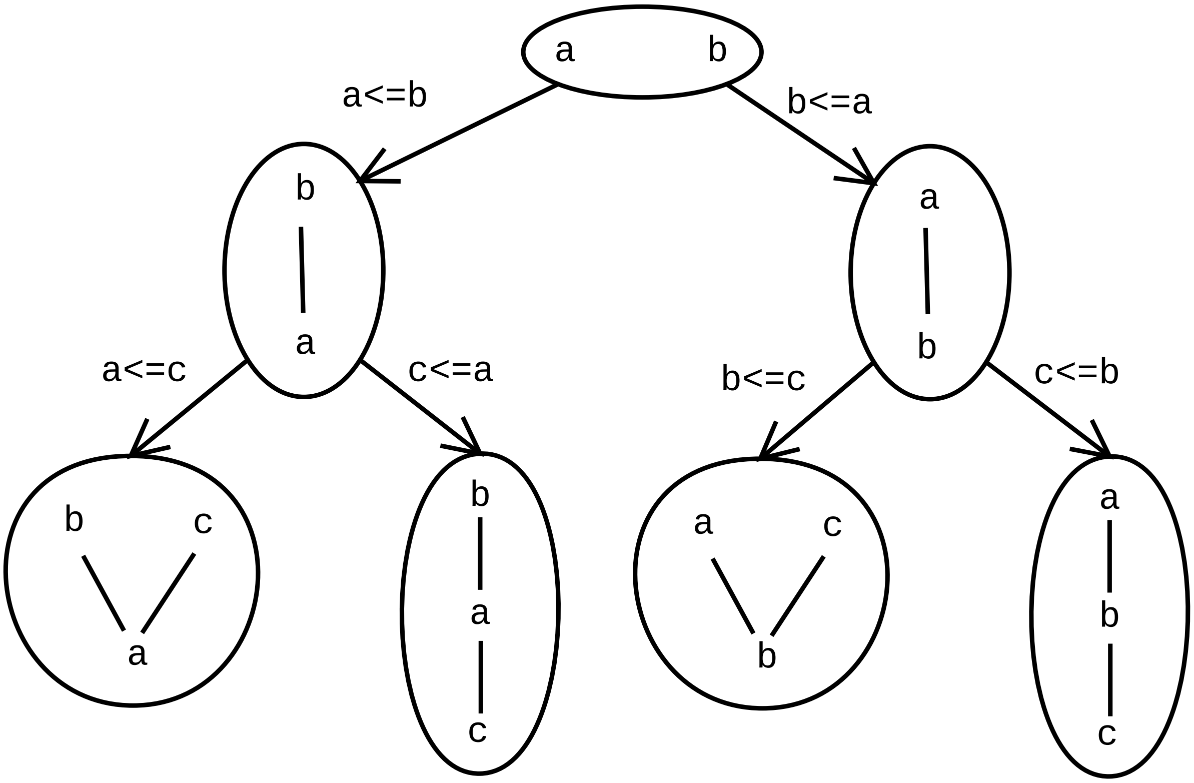
- Analysis: study the possible relationships between the data involved in the problem



Design

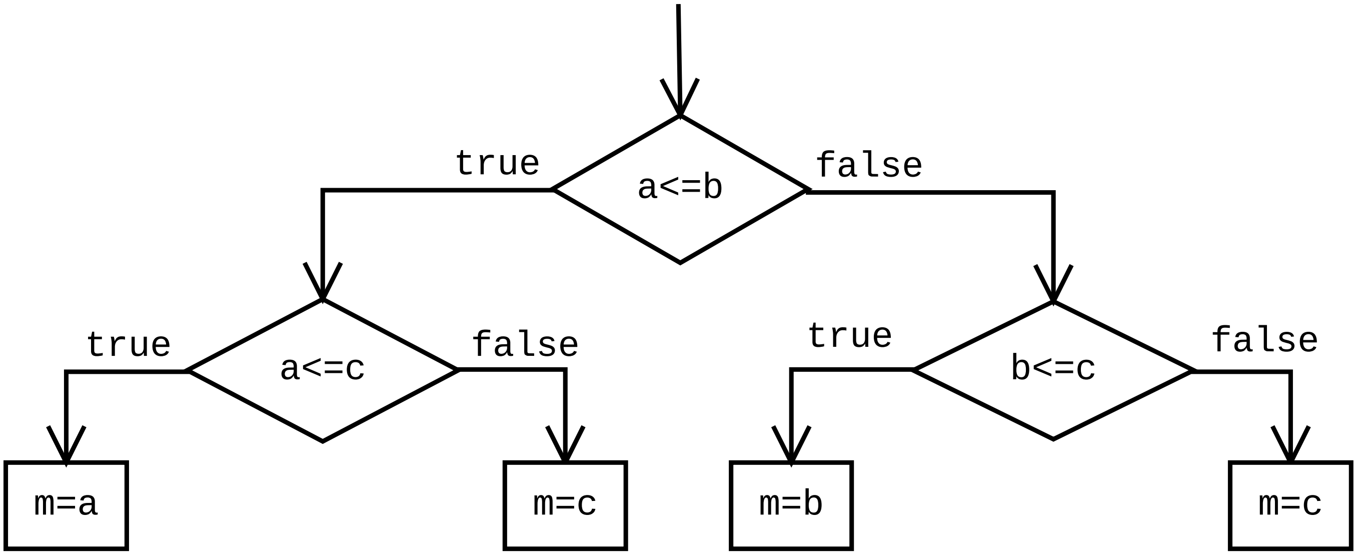


Design



Design

Decision tree



Implementation

```
import cs1.Keyboard;
public class SmallestFinder2 {
  public static void main(String[] args)
  {
    double a, b, c, m;

    System.out.print(‘‘Enter the first number:’’);
    a = Keyboard.readDouble();
    System.out.print(‘‘Enter the second number:’’);
    b = Keyboard.readDouble();
    System.out.print(‘‘Enter the third number:’’);
    c = Keyboard.readDouble();

    // Continues below ...
```

Implementation

```
    if (a <= b) {
      if (a <= c) {
        m = a;
      }
      else {
        m = c;
      }
    }
    else {
      if (b <= c) {
        m = b;
      }
      else {
        m = c;
      }
    }
    System.out.println(‘‘The smallest is ’’ + m);
  } // End of main method
} // End of SmallestFinder2 class
```

Properties of conditionals

- In the following, C is any boolean expression, P, Q, R, and S are any list of statements.

  ```
  P;
  if (C) {
    Q;
  }
  else {
    R;
  }
  S;
  ```

  is equivalent to

  ```
  P;
  if (!C) {
    R;
  }
  else {
    Q;
  }
  S;
  ```

Properties of conditionals

- In the following, C, D are any boolean expressions, P, Q, and R are any list of statements.

```
P;
if (C && D) {
  Q;
}
R;
```

is equivalent to

```
P;
if (C) {
  if (D) {
    Q;
  }
}
R;
```

Properties of conditionals

- In the following, `C`, `D` are any boolean expressions, `P`, `Q`, and `R` are any list of statements.

  ```
  P;
  if (C || D) {
    Q;
  }
  R;
  ```

  is equivalent to

  ```
  P;
  if (C) {
    Q;
  }
  else {
    if (D) {
       Q;
    }
  }
  R;
  ```

Data conversion

- Sometimes it is useful to look at data as if they were from a different type
- For example:
  - Adding an integer and a double
  - Obtaining the ASCII code of a character
- Forms of data conversion:
  - Implicit:
    - Assignment conversion
    - Promotion
  - Explicit: Casting

Data conversion

- Assignment conversion: A value of one type is assigned to a variable of a different type, as long as the types are compatible

```
int n = 7;
double d = n;
long k = n;
int m = d; // Wrong: compile-time error
```

- Promotion: an expression "promotes" the types of its operands to its "largest" type

```
int m = 8;
float x = 3.0f, y;
y = x + m;
```

Data conversion

- Casting expressions (not a statement)

  (*type*)*expression*

- Examples:

```
int n = 3;
double p;
p = (double)n + 4.0;

int a = 3, b = 8;
float c, d;
c = b/a;
d = (float)b/a;
System.out.println(c);  // 2.0
System.out.println(d);  // 2.666666...
```

Data conversion

```
double r = 2.41;
int a;
a = r; // Error
```

Data conversion

```
double r = 2.41;
int a;
a = (int)r;    //OK:  Narrowing casting
```

Data conversion

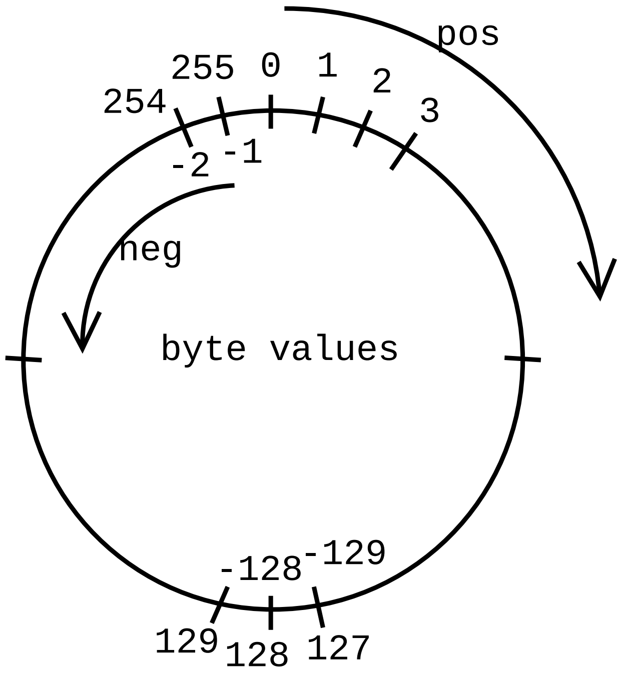
- There are two types of casting:
  - Narrowing conversions: from a type which requires more memory to a type that requires less
  - Widening conversions: from a type which requires less memory to a type which requires more
- If `expression` has type `t`, and `t` requires more memory than type `s`, then `(s)expression` is a narrowing conversion (e.g. `int` to `byte`, `double` to `float`, `float` to `int`, ...)
- If `expression` has type `t`, and `t` requires less memory than type `s`, then `(s)expression` is a widening conversion (e.g. `byte` to `double`, `long` to `int`, ...)

Data conversion

- Widening conversions are safe: no loss of information
- Narrowing conversions are not safe: possible loss of information

```
float x = 2.71f;
int i = (int)x;
// i == 2

int k = 130;
byte b = (byte)k;
// b = -126
```

Data conversion



128 = -128

129 = -127

256 = 0

257 = 1

byte b $\quad b+k2^8=b$

int i $\quad i+k2^{32}=i$

k is any integer

The end