Reminders

- Assignment 2 posted. Due on monday Oct 4th at 23:55 (late submission until wednesday, -5% per day)

Review

- Objects are composite data
- The data type of an object is a class
- The methods of a class are operations that can be applied to objects of that class (method invocation, or method call)

  objectref.*methodname*(*parameters*)

- Strings are objects
- String methods

```
int length()
char charAt(int index)
boolean equals(String s)
String substring(int initialIndex, int endIndex)
```

- If a method `m` returns a value of type `t`, then a method call to `m` can be used in any expression of type `t`

Class libraries

- Classes have a dual role:
  - as data types (of objects)
  - as modules
- Methods can be seen as "services" provided by these modules
- A *class library* is a collection of classes which has already been created by someone else, and which we can use in our programs
- Java has a Standard Library which includes classes for many different purposes
- Class libraries are organized in *packages*
- A *package* is a collection of classes

Class libraries

- Some packages from the Standard Java Library include:
  - `java.applet`: for creating programs which are easily transported accross the Web
  - `java.awt`: for creating GUIs (Graphical User Interfaces)
  - `java.io`: for handling files in secondary storage, or some communications
  - `java.lang`: general support
  - `java.math`: computations with high precision
  - `java.security`: support for security
  - `java.text`: for text manipulation
  - `java.util`: general utilities

- The full list of packages and classes in the Standard Library can be found at:

  `http://java.sun.com/j2se/1.4.2/docs/api/`

Class libraries

- Whenever we want to use a class `C` from a package `p`, we must use the import declaration at the top of the file:

  ```
  import p.C;
  ```

- For example, the `Keyboard` class is defined in the package `cs1`, so we must use

  ```
  import cs1.Keyboard;
  ```

  whenever we want to use it.

- Another example: the package java.util includes many useful classes such as `Currency` and `Calendar`. To use them we must add at the top of the file:

  ```
  import java.util.Currency;
  import java.util.Calendar;
  ```

Class libraries

- If we want to import all classes of a package `p`, we use

  ```
  import p.*;
  ```

- For instance:

  ```
  import java.util.*;
  ```

- The package `java.lang` is automatically imported in every program.

- Class `String` is defined in the `java.lang` package

- Class `Math` is also defined in the `java.lang` package

Static methods

- So far, all method calls that we have used take the form

  objectreference.*methodname*(*parameters*)

- But there are some methods that take the form

  classname.*methodname*(*parameters*)

- These are called *static methods*

- Static methods do not represent operations on objects, but services provided by a class

- For example:

```
i = Keyboard.readInt();
```

Static methods and class libraries

The Math class has many useful static methods, such as:

Method	Description
`static int abs(int num)`	returns the absolute value of `num`
`static double pow(double num, double power)`	returns $\texttt{num}^{\texttt{power}}$
`static double sqrt(double num)`	returns $\sqrt{\texttt{num}}$
`static double sin(double angle)`	returns $\sin(\texttt{angle})$
`static double cos(double angle)`	returns $\cos(\texttt{angle})$
`static double tan(double angle)`	returns $\tan(\texttt{angle})$
`static double floor(double num)`	returns the largest integer less or equal to `num`
`static double ceil(double num)`	returns the smallest integer greater or equal to `num`

Static methods and class libraries

```
double cathetus1, cathetus2, hypothenuse;
cathetus1 = 3.0;
cathetus2 = 4.0;
hypothenuse = Math.sqrt( Math.pow( cathetus1, 2 ) +
                         Math.pow( cathetus2, 2 ) );
```

Statements

- Variable declaration

```
type variable;
```

- Assignment

```
variable = expression;
```

- Method invocation

```
objectreference.methodname(parameters);
```

or

```
classname.methodname(parameters);
```

- Conditional

- Loop

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!");
}
```

Boolean expressions

Boolean operator	Meaning
&&	Logical conjunction (AND)
||	Logical disjunction (OR)
!	Logical negation (NOT)

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
}
```

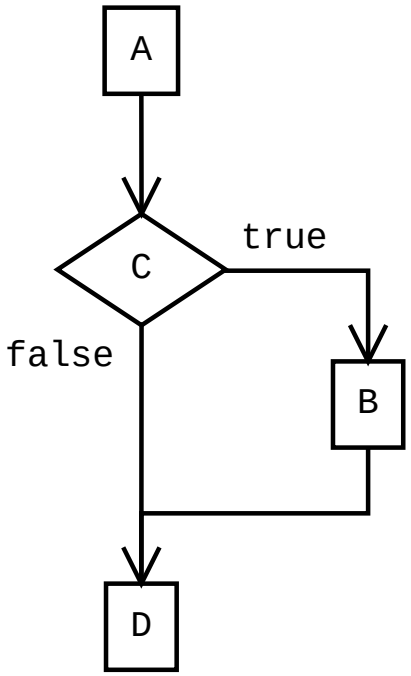
  is executed as follows:

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 = 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

- 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
}
```

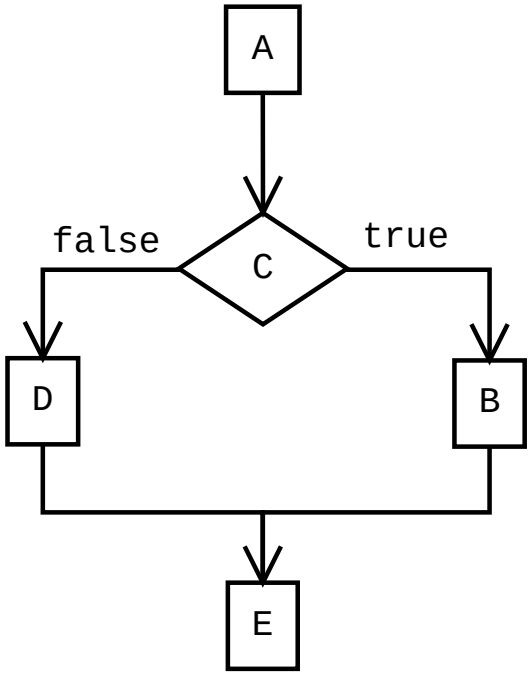
  is executed as follows:

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 = 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

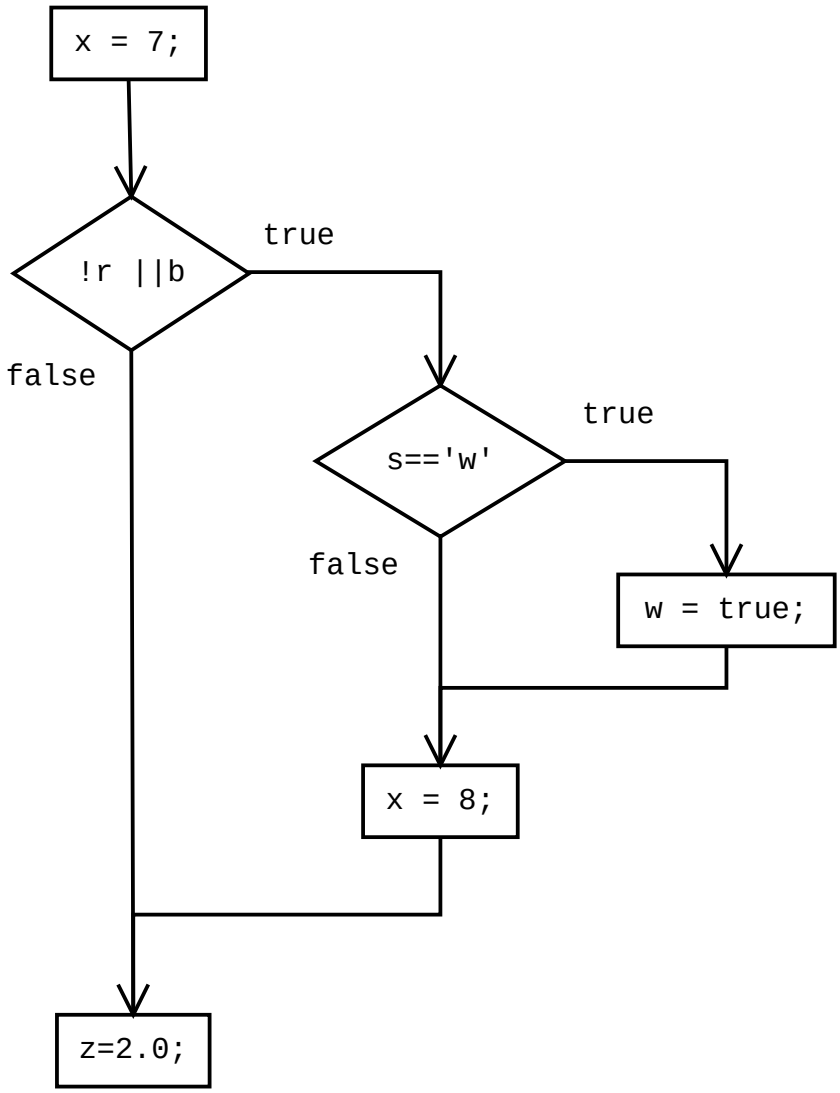
```
The house is vulnerable
The policeman eats donuts
```

Conditionals

- 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

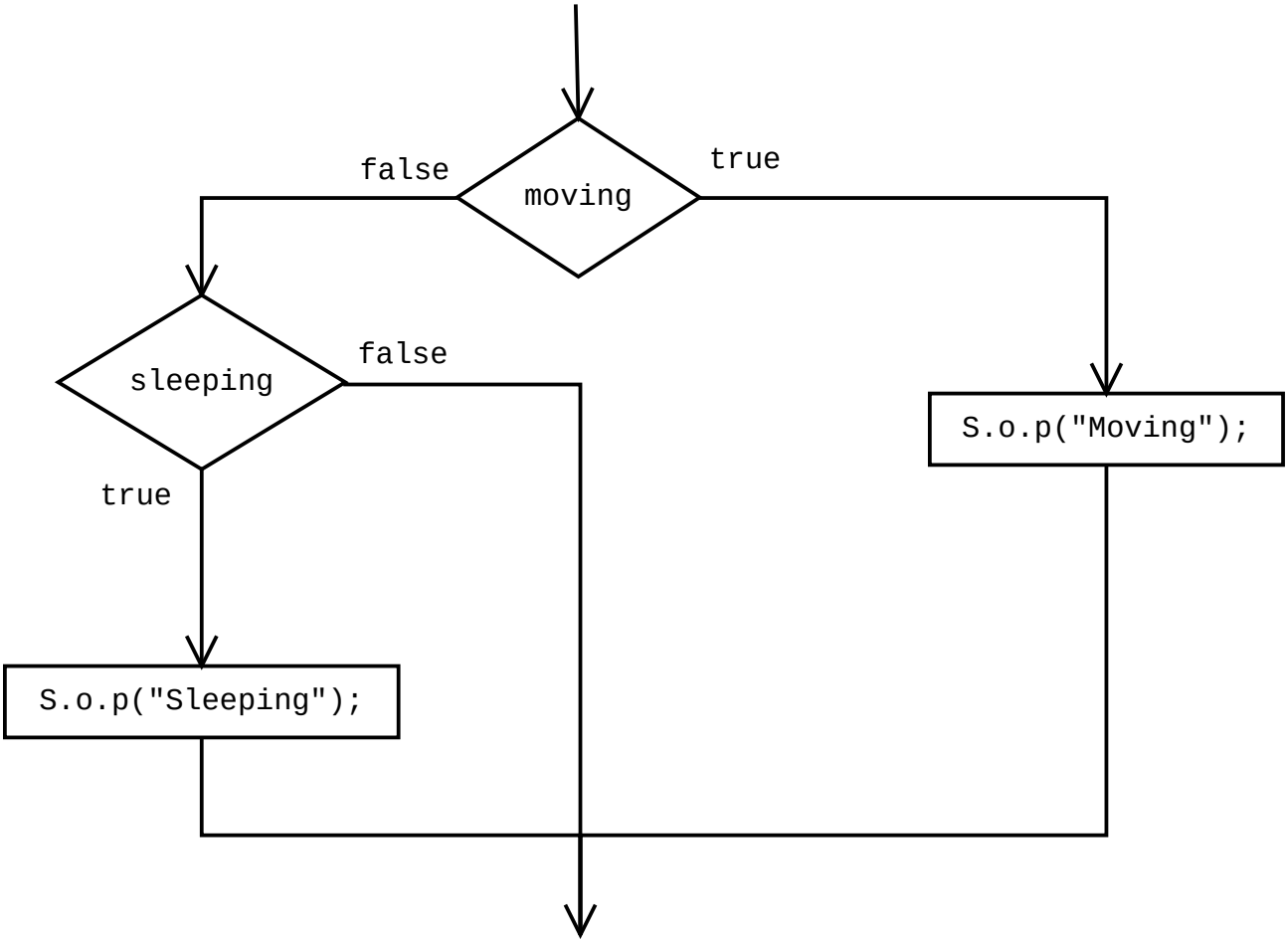


Conditionals

- 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”;
    }
}
```

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 “”
```