Statements

- Variable declaration

  ```
  type identifier;
  ```

- Assignment

  ```
  variable = expression;
  ```

- User Interface: output

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

- User Interface: input

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

Statements (contd.)

- Conditionals

  ```
  if (cond) { statements; }
  ```

  and

  ```
  if (cont) { stmts1; } else { stmts2; }
  ```

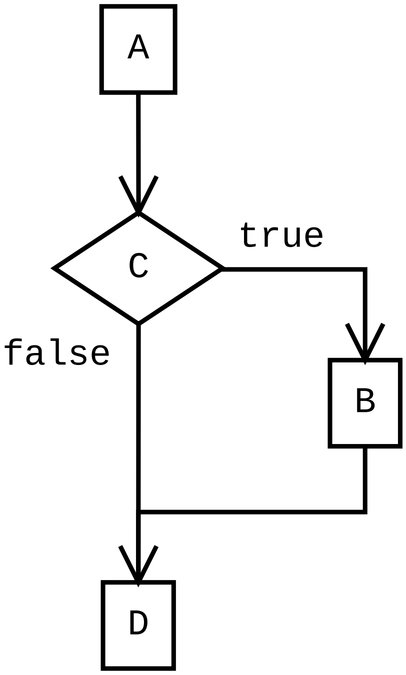
- Loops

- Method calls

Conditionals

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

- Control flow diagram



Conditionals

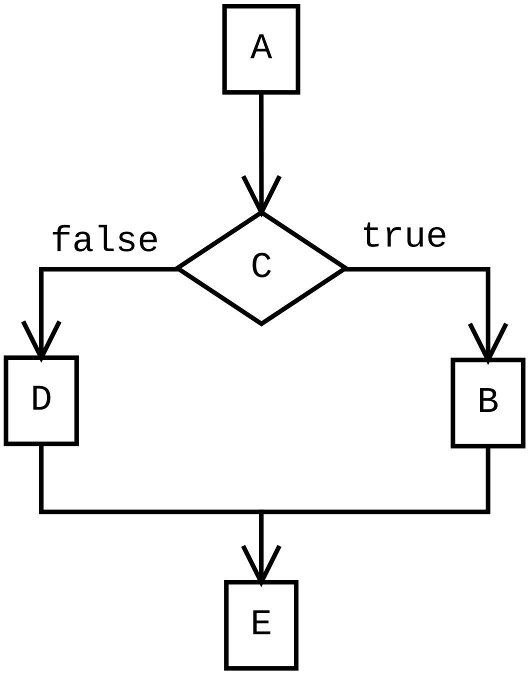
Example:

```
double temperature;
boolean windy, hat;
Scanner myScanner = new Scanner(System.in);

temperature = myScanner.nextDouble();
windy = true;
hat = false;

if (temperature < -20.0 && windy)
{
    hat = true;
}
System.out.println(hat);
```

Conditionals

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

- Control flow diagram



Conditionals

Example:

```
Scanner myScanner = new Scanner(System.in);
int x, y;
x = myScanner.nextInt();
y = 5;
if (x < 10)
{
    x = 15;
}
else
{
    y = x + 1;
}
System.out.println(x);
System.out.println(y);
```

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

- Consider the following:

```
int x = 4, y;
String z = “one”;
y = scanner.nextInt();
if (x > 3 && y < 6) {
  y = y + 8;
  z = “two”;
}
z = z + “three”;
```

is equivalent to

```
int x = 4, y;
String z = “one”;
y = scanner.nextInt();
if (x > 3) {
  if (y < 6) {
    y = y + 8;
    z = “two”;
  }
}
z = z + “three”;
```

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;
```

Properties of conditionals

- Consider the following:

```
boolean high = false;
double altitude;
altitude = scanner.nextDouble();
System.out.println(“Begin”);
if (altitude > 2000.0) {
  high = true;
  System.out.println(“It is high”);
}
else {
  high = true;
  System.out.println(“It is low”);
}
System.out.println(high);
```

Properties of conditionals

- It is equivalent to:

```
boolean high = false;
double altitude;
altitude = scanner.nextDouble();
System.out.println(“Begin”);
high = true;
if (altitude > 2000.0) {
  System.out.println(“It is high”);
}
else {
  System.out.println(“It is low”);
}
System.out.println(high);
```

Properties of conditionals

- Consider the following:

```
double altitude;
altitude = scanner.nextDouble();
System.out.println(‘‘Begin’’);
if (altitude > 2000.0) {
  altitude = altitude - 500.0;
  System.out.println(‘‘It is high’’);
}
else {
  altitude = altitude - 500.0;
  System.out.println(‘‘It is low’’);
}
System.out.println(altitude);
```

Properties of conditionals

- It is *not* equivalent to:

```
double altitude;
altitude = scanner.nextDouble();
System.out.println(“Begin”);
altitude = altitude - 500.0;
if (altitude > 2000.0) {
  System.out.println(“It is high”);
}
else {
  System.out.println(“It is low”);
}
System.out.println(altitude);
```

Properties of conditionals

- But it is equivalent to:

```
double altitude;
altitude = scanner.nextDouble();
System.out.println(‘‘Begin’’);
if (altitude > 2000.0) {
  System.out.println(‘‘It is high’’);
}
else {
  System.out.println(‘‘It is low’’);
}
altitude = altitude - 500.0;
System.out.println(altitude);
```

Properties of conditionals

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

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

Properties of conditionals

is equivalent to

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

if and only if the statements in `Q` do not modify the variables in `C`

Properties of conditionals

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

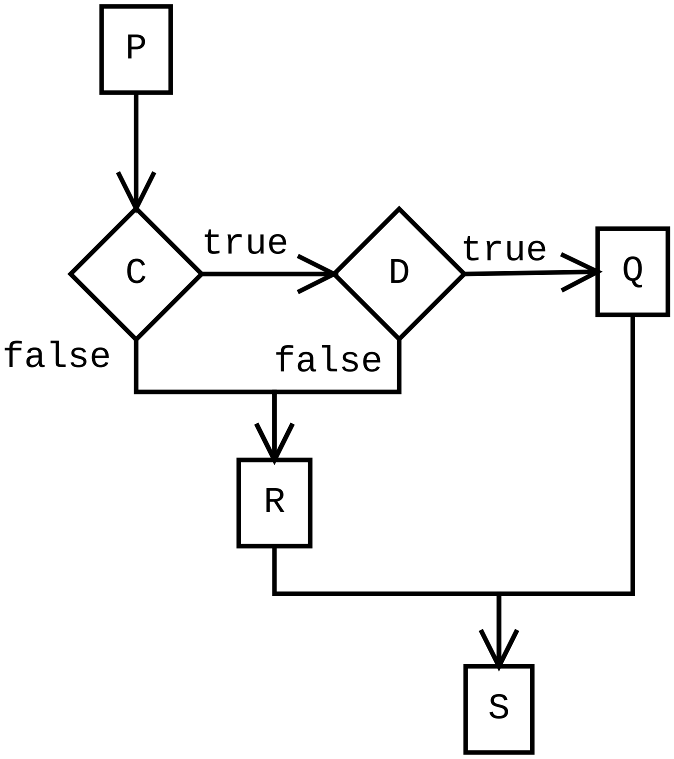
```
P;
if (C && D) {
  Q;
}
else {
  R;
}
S;
```

Properties of conditionals

is equivalent to

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

Properties of conditionals



Properties of conditionals

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

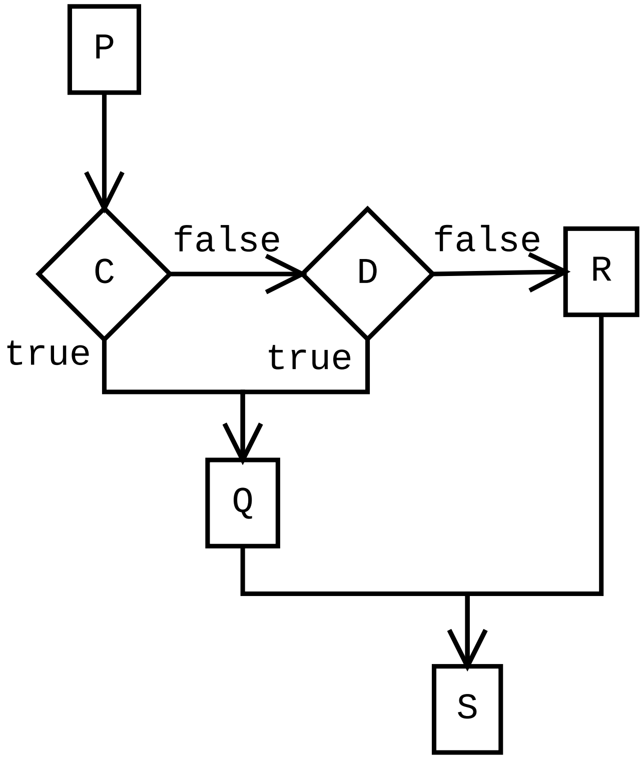
```
P;
if (C || D) {
  Q;
}
else {
  R;
}
S;
```

Properties of conditionals

is equivalent to

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

Properties of conditionals



Object-Oriented Programming

- Java is an *object-oriented* programming language
- The fundamental notion is that of an *object*
- Objects represent entities of a problem (possibly real-world entities)
- A program defines objects that interact with each other

Object-Oriented Programming

- Video games (RTS, FPS, RPG, etc.)
- Circuits
- Ecosystems
- ...

etc.

Objects and Classes

- Information in a Java program is represented by either
  - Primitive data (e.g. numbers, booleans)
  - Objects (composite data)

- An *object* is a composite piece of data which can be applied certain actions or operations:
  - An object is "made up" of other (simpler) pieces of data (primitive or objects)
  - An object is a group of data "glued" toghether that can be treated as a unit, a single piece of data
  - An object can "react" to operations we appy to it

Objects and Classes

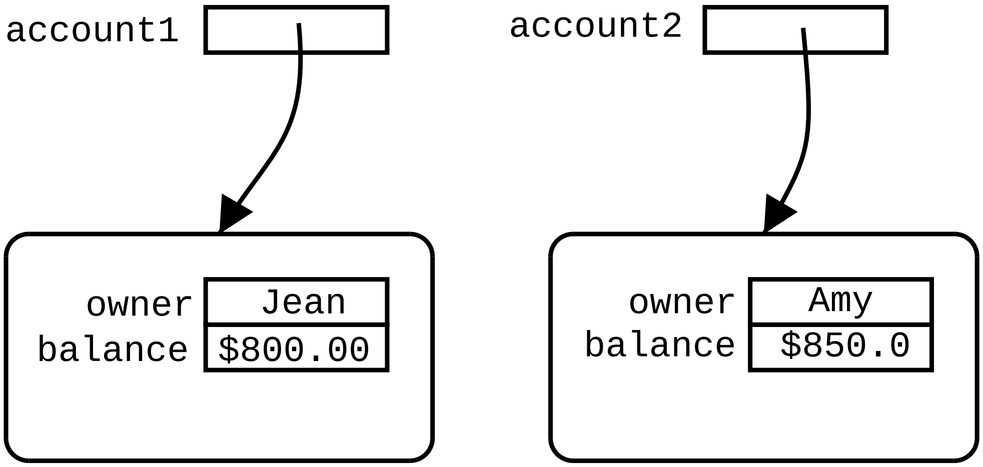
Example: a Stereo (physical object)

- Given a stereo we can:
  - Change the volume
  - Play a CD
  - Push FF (next song)
  - Rewind (previous song)
  - ...

Objects and Classes

- A bank account has:
  - owner
  - balance
- Given a bank account we can:
  - deposit
  - withdraw

Objects and Classes



Objects and Classes

- How do we create objects in our programs?

- Program structure:
  - A program is made up of classes
    - Classes are made up of methods
      - Methods are made up of statements

- What does this have to do with objects?

Objects and Classes

- Primitive data is defined by primitive data types (int, char, boolean)

- Objects are defined by Classes: the type of an object is a class

- Classes are given by a list of methods

- Methods: operations that can be performed on objects of the class where the method is defined

Objects and Classes

- Defining a class of objects

```
public class Stereo
{
    void change_volume()
    {
        // ...statements1...
    }

    void play()
    {
        // ...statements2...
    }

    void rewind()
    {
        // ...statements3...
    }

    //...
}
```

Objects and Classes

- Defining a class:

```
public class BankAccount
{
    String owner;
    double balance;

    void withdraw(double amount)
    {
      // ...
    }

    void deposit(double amount)
    {
      // ...
    }
}
```

- Note: only one class in a program has a `main` method

Using objects

- To be able to use objects we need:
  - Define some class or classes
  - A mechanism to create objects of a defined class
  - A mechanism to apply operations to these objects

Classes and Objects

1. Declare a variable of the appropriate type

2. Create an *instance* of the class we want

3. Call methods

Classes and Objects

- Declare a variable:

```
BankAccount account1;
```

- To *create objects* we use the `new` operator (with assignment)

```
account1 = new BankAccount();
```

- To *apply operations to objects* we use the *dot* operator:

```
account1.deposit(200.00);
```

- You cannot apply methods without first creating objects

Objects and Classes

- Defining a class:

```
public class BankAccount
{
    String owner;
    double balance;

    void withdraw(double amount)
    {
      // ...
    }

    void deposit(double amount)
    {
      // ...
    }
}
```

- Note: only one class in a program has a `main` method

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
    }
}
```

- Note: only one class in a program has a `main` method

Objects and Classes

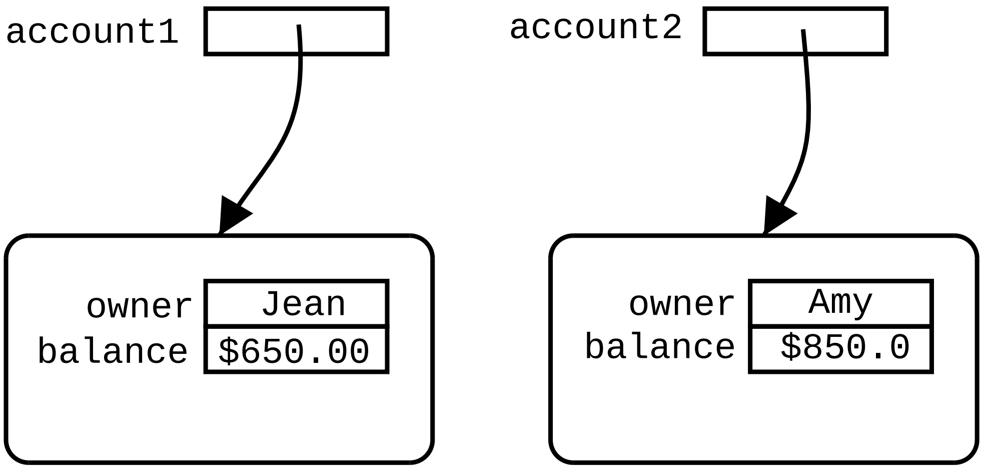
```
public class Test
{
    public static void main(String[] args)
    {
        BankAccount account1;
        account1 = new BankAccount();
        account1.deposit(200.0);
        BankAccount account2;
        account2 = new BankAccount();
        account2.deposit(150.0);
    }
}
```

- Each object has its own separate *identity*, its own individual *state*

Objects and Classes

$$\underbrace{\texttt{account1}}_{\text{object}} \;.\; \underbrace{\texttt{withdraw}}_{\text{method}} \; \underbrace{(150.00)}_{\text{parameters}};$$

- Applying a method to an object affects only the object it is being applied to.



$$\underbrace{\texttt{System.out}}_{\text{object}} \;.\underbrace{\texttt{println}}_{\text{method}} \; \underbrace{(\text{“}\texttt{text}\text{”})}_{\text{parameters}};$$

Scanner, Classes and Objects

```
int n;
Scanner myScanner;
myScanner = new Scanner(System.in);
n = myScanner.nextInt();
```

Strings, Classes and Objects

- Strings are objects
- The `String` type is a class
- To create String objects we can use the `new` operator

```
title = new String(‘‘Trainspotting’’);
```

- but this can be abbreviated as

```
title = ‘‘Trainspotting’’;
```

- ...only for Strings

Strings, Classes and Objects

- The String class has many methods

```
String title;
title = new String("Trainspotting");
title.toLowerCase();
```

- The statement

```
title.toLowerCase();
```

is a *method call* or *method invocation*

Strings, Classes and Objects

- Some methods of the String class
  - `charAt`: returns the character of the string at a given position
  - `length`: returns the length of the string
  - `equals`: returns whether the string is equal to another given string
  - `toLowerCase`: returns a copy of the string in lower case
  - `toUpperCase`: returns a copy of the string in uper case
  - `substring`: returns a part of the string given by the parameters
  - etc.

Strings, Classes and Objects

```
public class String {
  //...
  char charAt(int index) { //... }

  int length() { // ... }

  boolean equals(String str) { // ... }

  String toLowerCase() { //... }

  String toUpperCase() { //... }

  String substring(int offset, int endIndex) { //.

  //...
}
```

Strings, Classes and Objects

“	b	o	n	j	o	u	r	”
	0	1	2	3	4	5	6	

- In strings,
  - the first character has index 0
  - the second character has index 1
  - the third character has index 2
  - ...
  - the last character has index l-1, where l is the length of the string

Strings

- Examples of `int length()`

```
String question;
int l;
question = ‘‘Is this course  easy?’’;

l = question.length();

System.out.println(l);   // 21

String answer;
answer = ‘‘It depends...’’;

l = answer.length();

System.out.println(l);   // 13

String very_short_message = ‘‘’’;
System.out.println( very_short_message.length() );
```

Strings

- Examples of `char charAt(int index)`

```
String phrase;
char initial1, initial2, initial3,
     initial4, initial5;
String acronym;

phrase = ‘‘Emacs makes a computer swell’’;

initial1 = phrase.charAt(0);
initial2 = phrase.charAt(6);
initial3 = phrase.charAt(12);
initial4 = phrase.charAt(14);
initial5 = phrase.charAt(23);

acronym = ‘‘’’ + initial1 + initial2
               + initial3 + initial4 + initial5;
```

Strings

- The argument or parameter of `charAt` can be any integer expression

```
String phrase;
char c;
int start = 3;

phrase = “Strings do not have to make sense.”;

c = phrase.charAt( start + 2 );

// c == ‘g’

c = phrase.charAt( phrase.length() - 1 );

// c == ‘ .’

c = phrase.charAt( phrase.length() );
// Runtime error
```

Strings

- Since the `charAt` method returns a character, it can be used in any character expression, and in particular it can be used within string expressions

```
String word1 = “rat”, word2 = “case”;
String word3;
word3 = word1 + word2.charAt(2);

// word3 contains “rats”
```

Strings

- charAt cannot be used to modify a string

```
String word = “clap”;
word.charAt(0) = ‘f’; // WRONG!
```

- Strings in Java are immutable: they cannot change
- But String references can change:

```
String word = “clap”;
String new_word;
new_word = “f” + word.charAt(1)
         + word.charAt(2) + word.charAt(3);
word = new_word;

// word contains “flap”;
```

Strings

- Examples of `boolean equals(String s)`

  ```
  String pet1 = “cat”, pet2 = “rat”;
  String end1, end2;
  boolean same_pet, same_end;

  same_pet = pet1.equals(pet2);

  end1 = pet1.substring(1, pet1.length() );
  end2 = pet2.substring(1, pet2.length() );

  same_end = end1.equals(end2);
  ```

- For every pair of strings `a` and `b`, `a.equals(b)` returns the same as `b.equals(a)`

Strings

- Since the `equals` method returns a `boolean`, it can be used in any boolean expression

```
String season = ‘‘Winter’’;
float temp = -5.0f;
boolean warm;

warm = !season.equals(‘‘Winter’’) || temp >= -10.0f;
```

season.equals(‘‘Winter’’)	temp>=-10.0f	!season.equals("Wi
true	true	false
true	false	false
false	true	true
false	false	true

Strings

- Examples of

  `String substring(int offset, int endIndex)`

```
String word = “clap”;
String end, new_word;
end = word.substring(1, 4);

// end contains “lap”;

new_word = “f” + end;

// new_word contains “flap”
```

Strings

- `s.substring(i, j)` returns the part of string `s` beginning at index `i` and ending at index `j-1`

```
String phrase, subject, verb, article, noun;

phrase = “This is a string”;
subject = phrase.substring(0, 4);
verb = phrase.substring(5, 7);
article = phrase.substring(8, 9);
noun = phrase.substring(10, phrase.length());

System.out.println(subject+article+noun+verb);

// Prints
// Thisastringis
```

Strings

- Since the `substring` method returns a `String`, it can be used within any string expression

```
String old_phrase = “This is a string”;
int size = old_phrase.length();
String new_phrase;

new_phrase = old_phrase.substring(0, 8)
             + “not ”
             + old_phrase.substring(8, size);

// new_phrase contains “This is not a string”
```

Strings

- Examples of `String concat(String s)`

```
String sentence;
sentence = “This sentence is ”;
sentence = sentence.concat(“ false”);
```

- If `a` and `b` are strings, `a + b` is shorthand for `a.concat(b)`

Strings

- Examples of `String replace(char a, char b)`

```
String message, encoded;
message = “This message is irrelevant”;
encoded = message.replace(‘e’, ‘x’);

// encoded contains “This mxssagx is  irrxlxvant”

encoded = encoded.replace(‘a’, ‘y’);
encoded = encoded.replace(‘i’, ‘z’);
encoded = encoded.replace(‘r’, ‘w’);
encoded = encoded.replace(‘s’, ‘u’);
encoded = encoded.replace(‘ ’, ‘-’);
encoded = encoded.replace(‘t’, ‘v’);

// encoded contains “Thzu-mxuuygx-zu--zwwxlxvynv”
```

Strings, Classes and Objects

- Some method calls can appear as expressions and others as statements

```
String s = “abc”;
int n = s.length();
```

- Here, the call to method length is an expression because it occurs in the right-hand side of an assignment

```
s.toLowerCase(“abc”);
```

- Here, the call to the method toLowerCase is a statement because it is not being assigned to anything

The Random class

- Random number generation
- Random class methods

```
int nextInt()
int nextInt(int max)
float nextFloat()
double nextDouble()
```

- Must import the class from the `java.util` package

The Random class

```
import java.util.Random;
public class Test
{
  public static void main(String[] args)
  {
    Random generator;
    generator = new Random();
    int die;
    die = generator.nextInt(6);
  }
}
```

The Random class

```
Random generator;
generator = new Random();
int die;
die = generator.nextInt(6) + 1;
```

The Random class

```
Random generator;
generator = new Random();
int die;
die = generator.nextInt() % 6 + 1;
```

The Random class

```
Random generator;
generator = new Random();
int die;
die = generator.nextInt(6);
die = generator.nextInt(6);
```

Formatting numbers

- `NumberFormat` and `DecimalFormat` classes from the `java.text` package

- Methods

```
DecimalFormat(String pattern)
String format(double number)
void applyPattern(String pattern)
```

Formatting numbers

```java
import java.text.DecimalFormat;
public class Test
{
  public static void main(String[] args)
  {
    double n = 1.618314141;
    String output = “”;
    DecimalFormat formatter;
    formatter = new DecimalFormat();
    formatter.applyPattern(“0.###”);
    output = formatter.format(n);
    System.out.println(output);
  }
}
```

Formatting numbers

```
import java.text.DecimalFormat;
public class Test
{
  public static void main(String[] args)
  {
    double n = 1.618314141;
    String output = “”;
    DecimalFormat formatter;
    formatter = new DecimalFormat(“0.##”);
    output = formatter.format(n);
    System.out.println(output);
  }
}
```

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:

```
x = Math.sqrt(3);
```

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 $\mathtt{num}^{\mathtt{power}}$
`static double sqrt(double num)`	returns $\sqrt{\mathtt{num}}$
`static double sin(double angle)`	returns $\mathtt{sin}(\mathtt{angle})$
`static double cos(double angle)`	returns $\mathtt{cos}(\mathtt{angle})$
`static double tan(double angle)`	returns $\mathtt{tan}(\mathtt{angle})$
`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;
```

- Conditionals

```
if (cond) { statements; }
```

and

```
if (cont) { stmts1; } else { stmts2; }
```

- Loops
- Method invocation (aka method call)

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

or

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

Some shortcuts

```
x++;
```

means

```
x = x + 1;
```

```
x--;
```

means

```
x = x - 1;
```

```
x += 3;
```

means

```
x = x + 3;
```

Some shortcuts

- ++ and -- can be used inside arithmetic expressions (but it is not recommendable)

```
x = y-- * 2;
```

means:

```
x = y * 2;
y = y - 1;
```

and

```
x = --y * 2;
```

means

```
y = y - 1;
x = y * 2;
```

Characters

- Values of the `char` data type can be compared using the traditional relational operators:

```
char a = ’P’, b = ’Q’;
boolean c, d, e, f, g, h;
c = a == b;    // c == false
d = a != b;    // d == true
e = a < b;     // e == true
f = a > b;     // f == false
g = a <= b;    // g == true
h = a >= b;    // h == false

char a = ’Q’, b = ’Q’;
boolean c, d, e, f, g, h;
c = a == b;    // c == true
d = a != b;    // d == false
e = a < b;     // e == false
f = a > b;     // f == false
g = a <= b;    // g == true
h = a >= b;    // h == true
```

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`, ...)

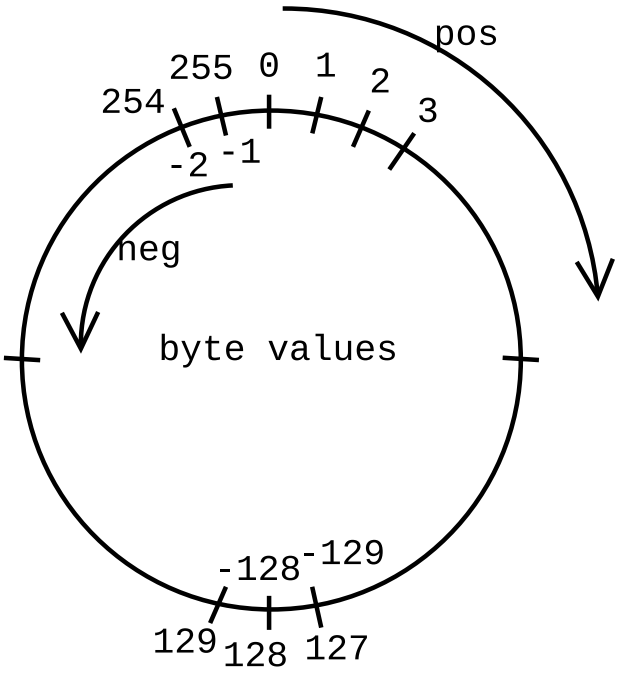
Data conversion

- 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	byte b	b+$k2^8$=b
129 = -127	int i	i+$k2^{32}$=i
256 = 0	k is any integer	
257 = 1		

The end