Statements

- Variable declaration

```
type identifier;
```

- Assignment

```
variable = expression;
```

- User Interface: output

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

- User Interface: input

```
variable = Keyboard.readType();
```

Assignment is not equality

```
x = a + b;
```

is an assignment statement which means:

1. evaluate `a+b`
2. and store the result in `x` (overwriting whatever `x` had before)

by contrast

```
x == a + b
```

is a boolean expression which has a truth value depending on the values of `x`, `a` and `b`.

```
int x, y;
x = 4;
y = 7;
x = y + 2;
```

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

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`, ...)

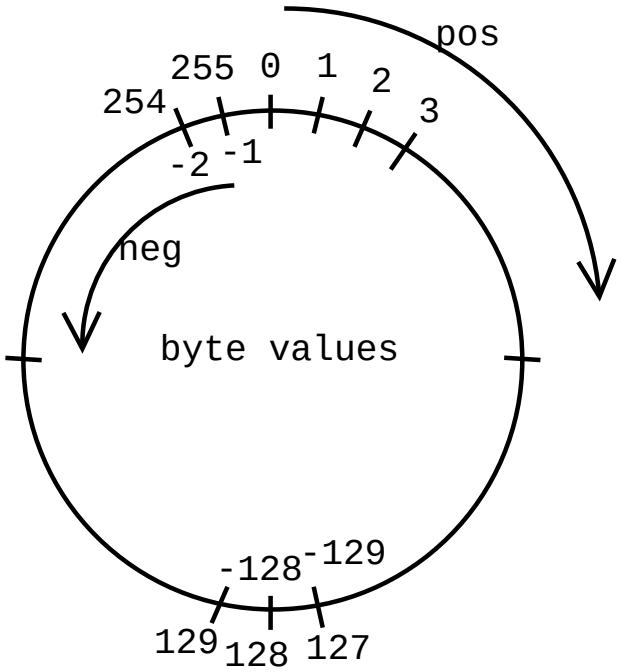
Data conversion

- 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		

Precedence

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

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;
```

Objects and Classes

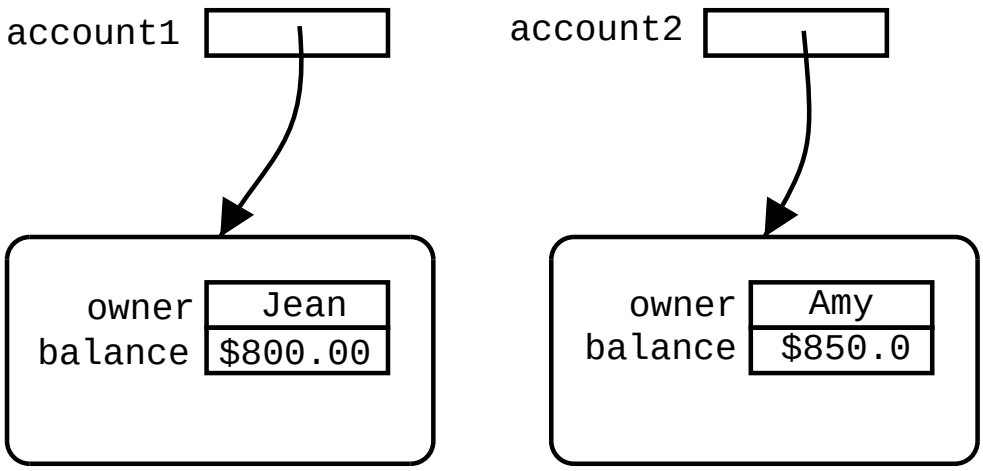
- Information in a Java program is represented by either
  - Primitive data (e.g. numbers, booleans)
  - Objects (composite data)

- 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 a s a unit, a single piece of data
  - An object can "react" to operations we appy to it

Objects and Classes

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

Objects and Classes



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
- 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

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

Classes and Objects

- A variable contains either
  - a primitive value (e.g. 2, 3.14, true, ...)
  - or a reference to an object
- Declaring a variable of a non-primitive type does not create an object of that type:

```
BankAccount account1;
```

- This declaration only results in allocating (reserving) a memory cell which may hold a reference to a BankAccount object

Classes and Objects

- To *create objects* we use the `new` operator

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

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

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

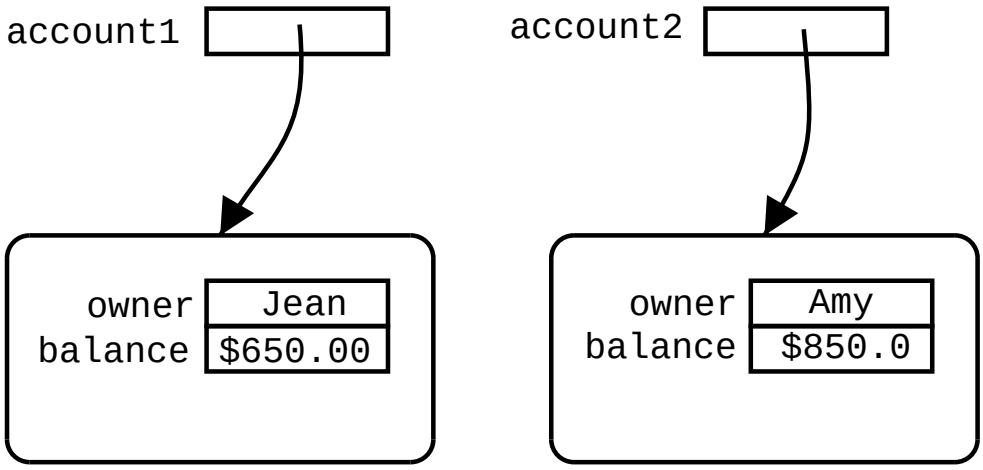
- You cannot apply methods without first creating objects

Objects and Classes

```
account1 . withdraw (150.00);
```

object method parameters

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




```
System.out .println ("text");
```

object method parameters

Strings, Classes and Objects

- Strings are objects
- The String type is a class, predifined in the *Standard Java Library*
- A *class library* is a collection of predefined classes
- 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
  - `toLowerCase`: returns a copy of the string in lower case
  - `toUpperCase`: returns a copy of the string in uper case
  - `equals`: returns whether the string is equal to another given string
  - `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() { // ... }

  String toLowerCase() { //... }

  String toUpperCase() { //... }

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

  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
  `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 `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("Winter")	warm
true	true	false	true
true	false	false	false
false	true	true	true
false	false	true	true

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

Strings, Classes and Objects

```
String name, first_name, last_name;
char initial1, initial2;
int len;

name = new String(“Charles Darwin”);
initial1 = name.charAt(0);
initial2 = name.charAt(8);
len = name.length();
first_name = name.substring(0, 7);
last_name = name.substring(8, len);

System.out.println(first_name);
System.out.println(last_name);
System.out.println(“”+initial1+initial2);
```

Strings, Classes and Objects

```
Charles
Darwin
CD
```

An example

- Problem: Given a four letter word, print the word in reverse.

- Analysis:
  - Information involved: a four letter word, w.
  - Input: w
  - Output: a word v which is the reverse of w
  - Definitions:
    * The *reverse* of a word w is a word v which has the the same characters as w, but in inverse order: the first letter of v is the last of w, the second letter of v is the second-to-last of w, etc.
  - Restrictions: w is assumed to have only four letters

An example

- Design

1. Obtain the word w
2. Create a new word v, initially empty
3. Add the last character of w to the end of v
4. Add the third character of w to the end of v
5. Add the second character of w to the end of v
6. Add the first character of w to the end of v
7. Print v

An example

- Implementation

```
import cs1.Keyboard;
public class Reverse {
  public static void main(String[] args)
  {
    String w, v;

    System.out.print(“Enter a four letter word: ”)
    w = Keyboard.readString();

    v = ““;
    v = v + w.charAt( 3 );
    v = v + w.charAt( 2 );
    v = v + w.charAt( 1 );
    v = v + w.charAt( 0 );

    System.out.println(v);
  }
}
```

An example

- Implementation

```
import cs1.Keyboard;
public class Reverse {
  public static void main(String[] args)
  {
    String w, v;

    System.out.print("Enter a four letter word: ")
    w = Keyboard.readString();

    v = "" + w.charAt( 3 ) + w.charAt( 2 )
        + w.charAt( 1 ) + w.charAt( 0 );

    System.out.println(v);
  }
}
```

An example

- Design (alternative)

1. Obtain the word w
2. Create a new word v, initially empty
3. Add the first character of w to the front of v
4. Add the second character of w to the front of v
5. Add the third character of w to the front of v
6. Add the last character of w to the front of v
7. Print v

An example

- Implementation

```
import cs1.Keyboard;
public class Reverse {
  public static void main(String[] args)
  {
    String w, v;

    System.out.print(“Enter a four letter word: ”)
    w = Keyboard.readString();

    v = ““;
    v = ““ + w.charAt( 0 ) + v;
    v = ““ + w.charAt( 1 ) + v;
    v = ““ + w.charAt( 2 ) + v;
    v = ““ + w.charAt( 3 ) + v;

    System.out.println(v);
  }
}
```

An example

- Implementation

```
import cs1.Keyboard;
public class Reverse {
  public static void main(String[] args)
  {
    String w, v;

    System.out.print("Enter a four letter word: ")
    w = Keyboard.readString();

    v = "" + w.charAt( 3 ) + w.charAt( 2 )
        + w.charAt( 1 ) + w.charAt( 0 );

    System.out.println(v);
  }
}
```

Another example

- Problem: Given a four letter word, determine whether the word is a palindrome

- Analysis:
  - Information involved: a four letter word, w.
  - Input: w
  - Output: true if the word is a palindrome, false otherwise
  - Definitions:
    * A word is a *palindrome* if it is the same as its own reverse, e.g. (noon, radar, wow, pop, 2002, ...)
  - Restrictions: w is assumed to have only four letters

Another example

- Design:

1. Obtain word *w*
2. Compute the reverse of *w*: let *v* be the reverse of *w*
3. Compare *v* and *w*. Let *result* be true if *w* and *v* are equal, and false otherwise.
4. Print *result*

Another example

```
import cs1.Keyboard;
public class Palindromes {
  public static void main(String[] args)
  {
    String w, v;
    boolean result;

    System.out.print(“Enter a four letter word: ”)
    w = Keyboard.readString();

    v = ““;
    v = v + w.charAt(3);
    v = v + w.charAt(2);
    v = v + w.charAt(1);
    v = v + w.charAt(0);

    result = v.equals(w);

    System.out.println(result);
  }
}
```

Another example

- Design (alternative):

1. Obtain word *w*

2. Compare the first character of *w* with its last character and the second character with the thirs character. Let result be true if both comparisons yield true, and false otherwise.

3. Print *result*

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
```

Another example

```
import cs1.Keyboard;
public class Palindromes {
  public static void main(String[] args)
  {
    String w;
    boolean result;

    System.out.print(“Enter a four letter word: ”)
    w = Keyboard.readString();

    result = w.charAt(0) == w.charAt(3)
           && w.charAt(1) == w.charAt(2);

    System.out.println(result);
  }
}
```