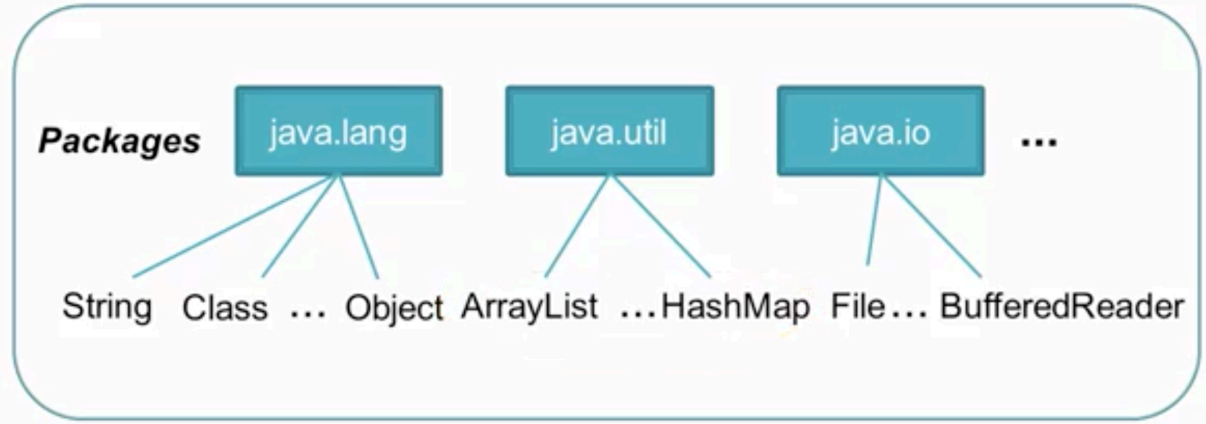
Java API

- Library of *well-tested* classes
- Java 8 ~ **4240** classes
- Developed by experts
- Used by *millions* of programmers
- Part of both JDK & JRE

Packages



Why Packages?

- Meaningful organization
- Name scoping

  java.util.Date **!=** java.sql.Date
- Security

Java API: Important Packages

- java.lang ~ Fundamental classes
- java.util ~ Data structures
- java.io ~ Reading & writing
- java.net ~ Networking
- java.sql ~ Databases

API Benefits

- Focus on writing new logic
- APIs *improve performance* over time
- Gain new functionality too

Accessing Classes

- Same package ~ *direct* access
- Different package
  - ***import***
  - Fully-qualified class name ~ *rare!*

import Statement

```
import java.util.ArrayList;

class FooClass {
  void foo() {
    ArrayList list = new ArrayList();
    …
  }
}
```

Importing Single vs Multiple Classes

- Import *single* class
  - Explicit import (or *Single-Type-Import*)
- Import *multiple* classes
  - Separate explicit import
  - * import (or *Import-On-Demand*)

* import

Imports *all* classes in a package

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

select * from <table_name>

Explicit import or * import?

- * import can ***break*** code

```
import java.util.*;                          import java.util.*;
import java.sql.*;      + java.sql.Date      import java.sql.*;
…                                            …
Date date; // from util                      Date date; // compiler error
```

- *Better clarity* with explicit import
- Explicit import seems to be preferred

Fully-qualified Class Name

- Alternative to *import*

```
java.util.ArrayList list = new java.util.ArrayList();
```

- Required if using *java.util.Date* & *java.sql.Date*

java.lang is imported by ***default***

Solution 1

Use only **one explicit** import

```
import java.util.Date;
import java.sql.*;

…

Date date; // from util
java.sql.Date date2;
```

Solution 2

Use **only fully-qualified** names

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

…
java.util.Date date;
java.sql.Date date2;
```

Invalid Imports

```
import java.util.Date;
import java.sql.Date;
```

Any Side Affects in Using import?

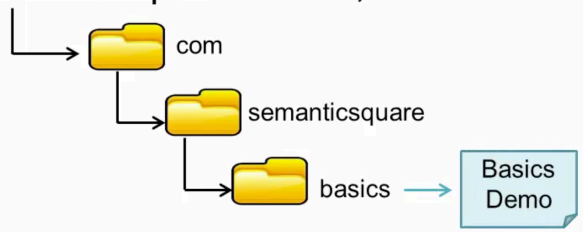
Nope!!

- *Does not make your class bigger!!*
- *Does not affect runtime performance*
- Saves from typing fully-qualified name ~ compiler does this

Set-up Matching Directory Structure

- basics;

- com.semanticsquare.basics;



package Statement

- **package** package-name

```
package com.semanticsquare.basics;
```

- Must be ***first*** statement above any imports

```
package com.semanticsquare.basics;
import java.util.ArrayList;
class BasicsDemo {
   …
}
```

- ✓ Ensure *matching directory structure* exists
- ✓ Use *package* statement

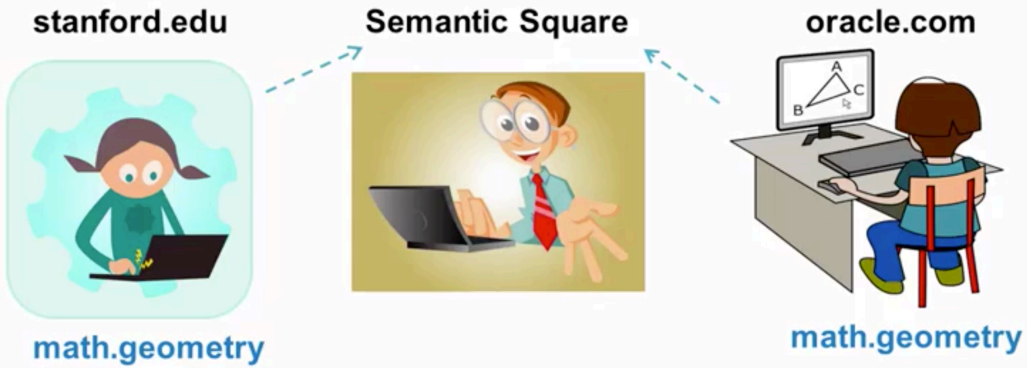
Package name is part of class name

java *BasicsDemo* ~ *will not work*
java *com.semanticsquare.basics.BasicsDemo*

Sub-packages

- *java.util* & *java.util.concurrent*
- java.util.* ~ imports *only* util
- java.util.*.* ~ **invalid**

Package Naming



package name conflict

Avoiding Package Name Conflicts

Use organization's *reverse internet domain name*

edu.stanford.math.geometry
com.oracle.math.geometry

Components Naming Conventions

- **Lowercase alphabets**, *rarely digits*
- Short ~ generally, *less than **8*** characters
- Meaningful abbreviations, e.g., util for utilities
- Acronyms are fine, e.g., awt for Abstract Window Toolkit
- Generally, single word
- Never start with *java* or *javax*

Strings

Object of class ***java.lang.String***

```
String s = new String(); // empty string
String s = new String("hello!");

char[] cArray = {'h', 'e', 'l', 'l', 'o', '!'};
String s = new String(cArray);

String s = "hello!"; // string literal    (recommended)
```

Strings

- String class uses **character array** to store text
- Java uses **UTF-16** for characters
- String is *sequence of unicode characters*
- String is *immutable*

String object ~ *immutable sequence of unicode characters*

String is Special

- String literal
- **+** operator

```
String s = “hello” + “ world!”; // “hello world!”
```

- String pool ~ *saves memory*

3rd Party String Utilities

- Apache Commons Lang ~ **StringUtils**
- **Guava’s** String Utility Classes

Common Operations

- Comparing
- Searching
- Examining individual characters
- Extracting substrings
- Case translation
- Replace
- Split

String Literal vs Using new

- String (via *string literal*)
  - Stored in *string pool* on heap
  - Literals with same content *share storage*
- String (via *new*)
  - Same as regular object
  - No storage sharing

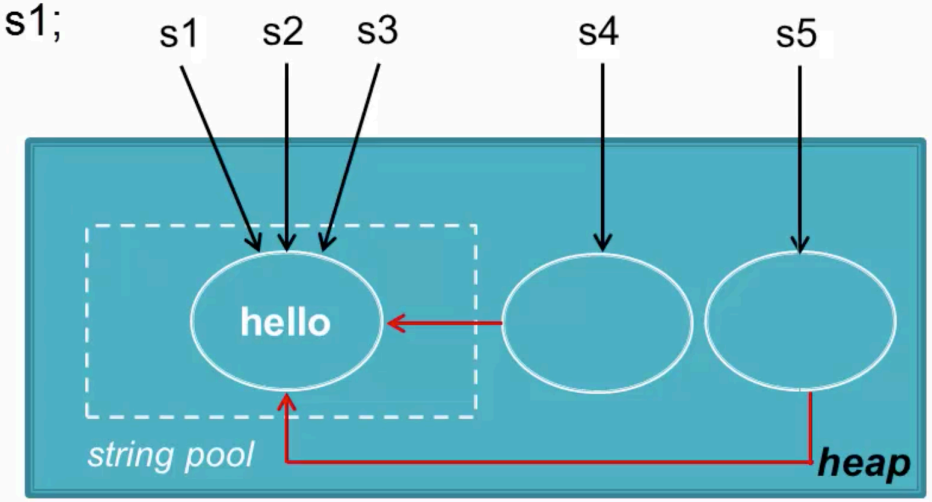
String *s1* = "hello";

String *s2* = "hello";

String *s3* = s1;

String *s4* = **new** String("hello");

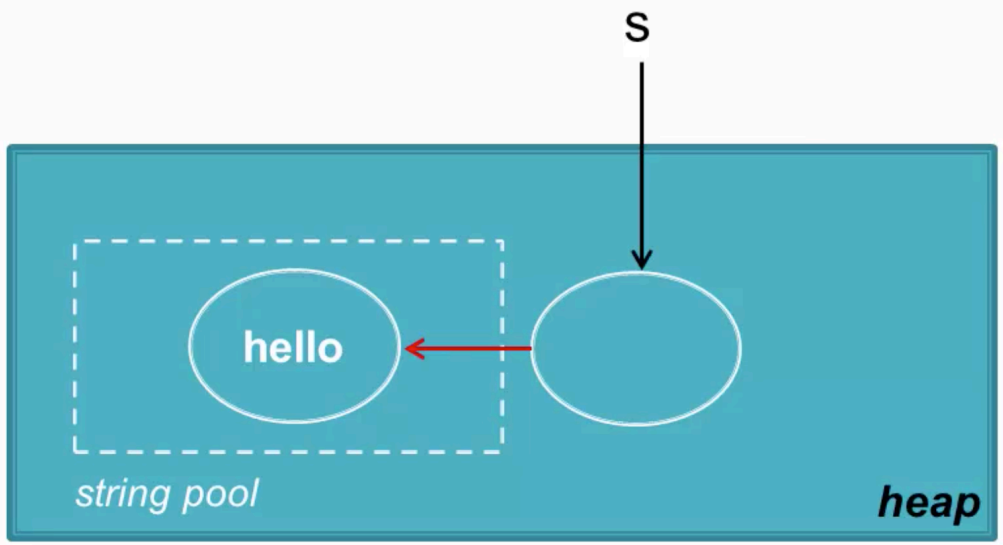
String *s5* = **new** String("hello");



s1 == s2? True

s4 == s5? False

String *s* = **new** String("hello");



String Pool

- Stores ***single*** copy of each string literal as **string object**
- Only ***one*** string pool
- Also called **string table**

String Interning by JVM

encountering a string literal for first time

✓ Create new **String** object with given literal

✓ Invoke ***intern***()

if (string *in* **string pool**)
  return *existing* reference
else
  add to string pool and return reference

Few Examples

- String *s* = “hel” + “lo”; ~ *interned* too
- String *s1* = “lo”;
  String *s2* = “hel” + s1; ~ *not interned*
  s2 = s2.***intern***(); ~ *explicit interning*

Is explicit interning useful?

Most likely not. Mostly for JVM

E.g., Natural language processing (NLP) ~ *needs benchmarking!!*

String Immutability

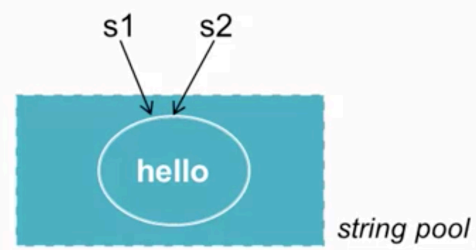
Value can *never* be changed

String s1 = new String(“abcd”); // or “abcd”
s1 = new String(“1234”); // above object is abandoned

Why Immutability?

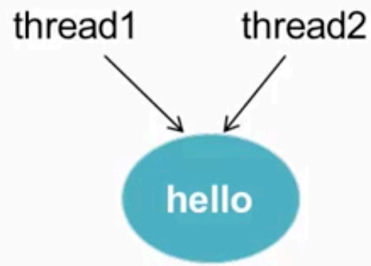
1. String interning

If mutable, *sharing* is not possible



Why Immutability?

2. Concurrency



Why Immutability?

3. Security

FileInputStream(String name)

String Concatenation

+ operator

String s = “hello” **+** “ world!”;

String s = “hello” + “ world!” + “125”; ~ *“hello world!125”*

String s = “hello” + “world!” + 125;

String s = “hello” + “world!” + 125 + 25.5 ~ *“hello world!12525.5”*

String s = 125 + 25.5 + “hello” + “world!” ~ “150.5hello world!”

String Concatenation

- StringBuilder
- StringBuffer

StringBuilder

- From Java 5
- Example

```
StringBuilder sb = new StringBuilder();
sb.append("hello");
sb.append(" world!");
String s = sb.append(" Good").append(" morning").toString();
```

- Other methods: *length*, *delete*, *insert*, *reverse*, *replace*
- Not synchronized

StringBuffer

- Obsolete. Use *StringBuilder!*
- Synchronized ~ slow
- API compatible with StringBuilder

Beware the performance of string concatenation

Item 51: + Operator

- Combining few strings is fine
- With each concatenation,
  - Contents of both strings are ***copied***
  - New **StringBuilder** is created and appended with both strings
  - Return string via toString()

Item 51: + Operator Example

Concatenating *a*, *b*, *c* in a loop

```
s += “a”; // copy of “” & a are made to generate a
s += “b”; // copy of a & b are made to generate ab
s += “c”; // copy of ab & c are made to generate abc
```

Also, StringBuilder is created for each concatenation

Time consuming ~ **$O(N^2)$**, Space consuming

Item 51: Use StringBuilder

- **O(N)**
- A/C one benchmark,
  - StringBuilder = ***300x*** times **+** operator
  - StringBuilder = ***2x*** times StringBuffer

Escape Sequence

- Character preceded by \
- To use ***special characters*** in strings & character literals

Escape Sequences

- \" ~ double quote *(not required in char literal)*
- \' ~ single quote *(not required in String literal)*
- \n ~ new line
- \t ~ tab
- \\ ~ backslash
- \r ~ carriage return
- \b ~ backspace
- \f ~ formfeed

char c = '\u0041';

Accessibility for Classes/Interfaces

- Inside package
- Inside & outside package ~ **public**

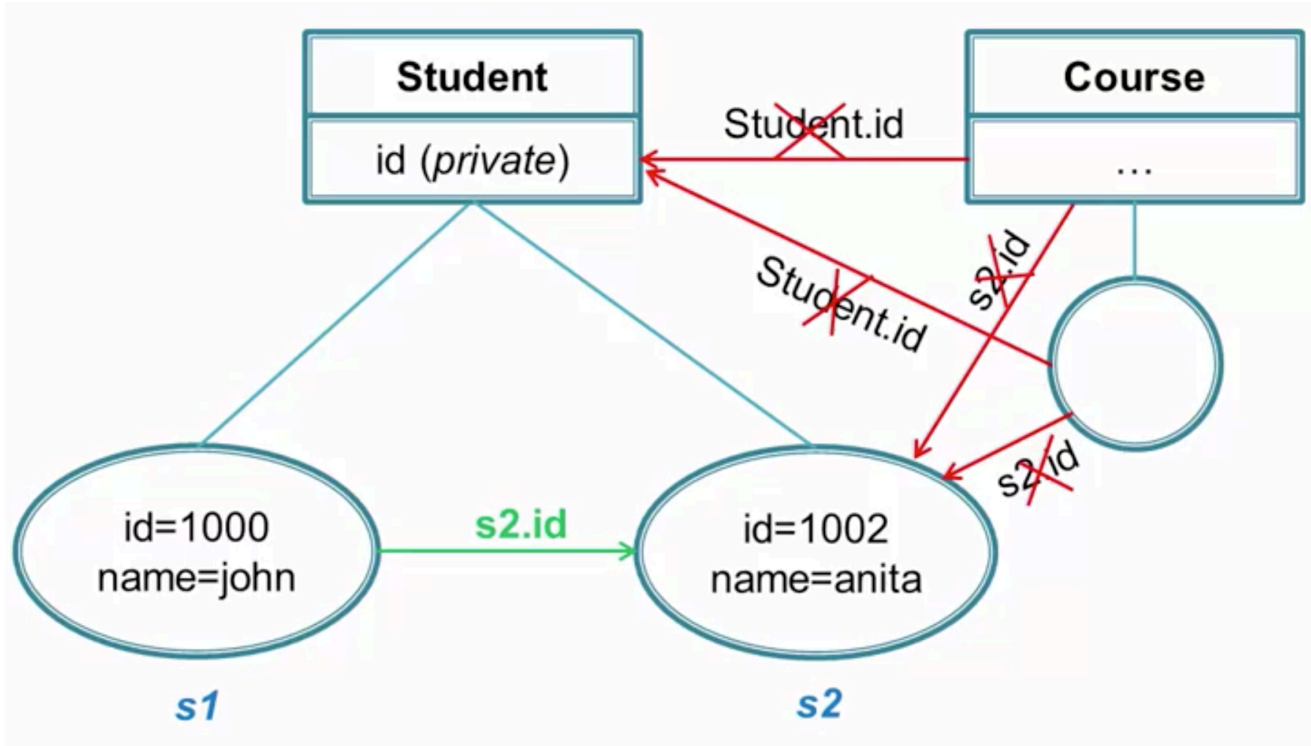
```
public class BasicsDemo {
    …
}
```

private Access Modifier

- **private** int id;
- **private** void go() { … }
- Private means *private to class and not object*

Accessibility for Class Members

- Inside class ~ **private**
- Inside package
- Inside package + *any* subclass ~ **protected**
- Inside & outside package ~ **public**


Student
id (private)
Course
...
Student.id
Student.id
s2.id
s2.id
id=1000
name=john
s2.id
id=1002
name=anita
s1
s2