University of British Columbia
CPSC 111, Intro to Computation
2009W2: Jan-Apr 2010

Tamara Munzner

Objects, Strings, Parameters

Lecture 6, Mon Jan 18 2010

borrowing from slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/111-10

News

- CS dept announcements

- Undergraduate Summer Research Award (USRA)
  - applications due Feb 26
  - see Guiliana for more details

Department of Computer Science
Undergraduate Events

Events this week

Drop-in Resume/Cover Letter Editing

Date: Tues., Jan 19
Time: 12:30 – 2 pm
Location: Rm 255, ICICS/CS Bldg.

Interview Skills Workshop

Date: Thurs., Jan 21
Time: 12:30 – 2 pm
Location: DMP 201
Registration: Email dianejoh@cs.ubc.ca

Project Management Workshop

Speaker: David Hunter (ex-VP, SAP)
Date: Thurs., Jan 21
Time: 5:30 – 7 pm
Location: DMP 110

CSSS Laser Tag

Date: Sun., Jan 24
Time: 7 – 9 pm
Location: Planet Laser @ 100 Braid St., New Westminster

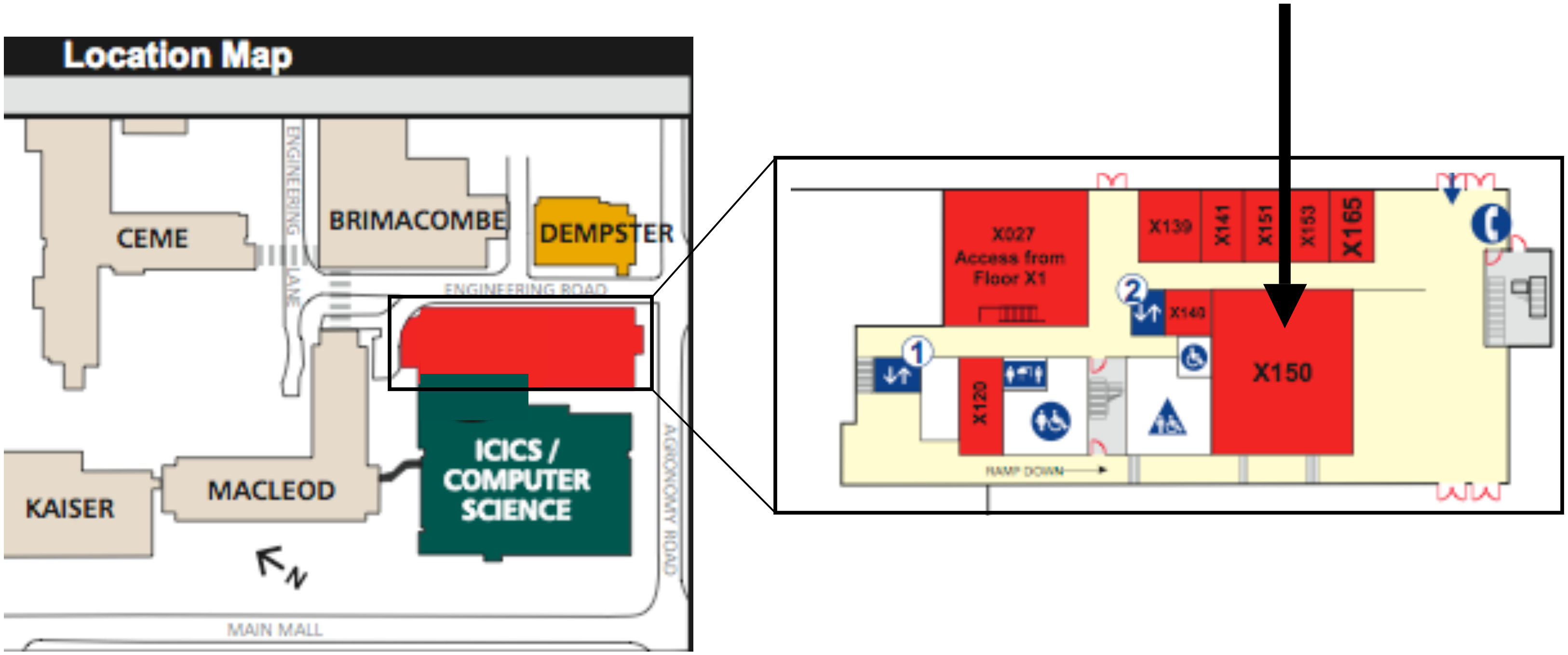
Event next week

Public Speaking 101

Date: Mon., Jan 25
Time: 5 – 6 pm
Location: DMP 101

Resources

- Demco Learning Center: drop by if you have any questions!
  - ICICS/CS x150
  - Normal schedule starts today
    - 10 am - 6 pm M-Th, 10 am - 4 pm F
    - Staffed by TAs from all 1st year courses, see schedule at http://www.cs.ubc.ca/ugrad/current/resources/cslearning.shtml



More Resources

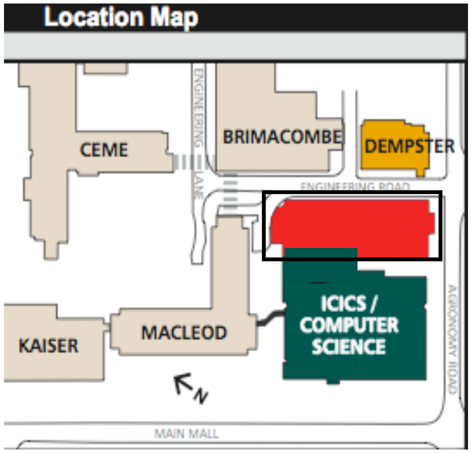
- WebCT discussion groups
  - Monitored by TAs/instructor, use to ask questions

- don’t forget to check web page first/often!
  - lecture slides, handouts, schedule, links, ...
  - http://www.cs.ubc.ca/~tmm/courses/111-10

Yet More Resources

- reminder: my office hours Mondays 4-5pm, starting today

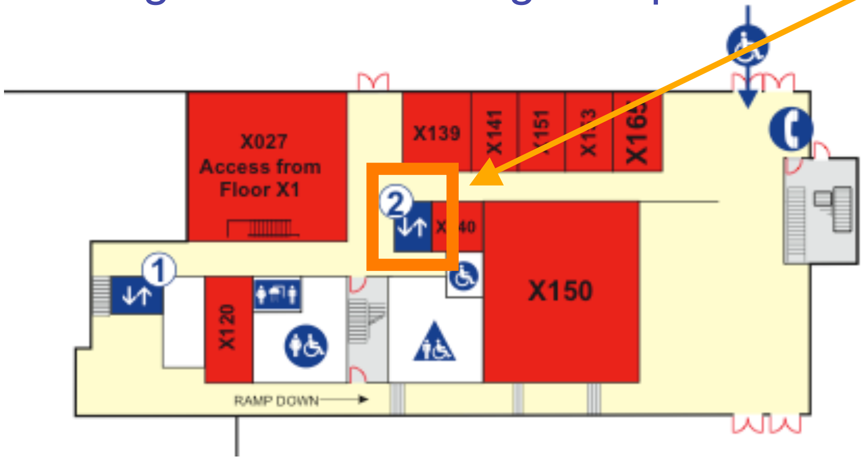
- office location is X661 (tall wing of ICICS/CS bldg)

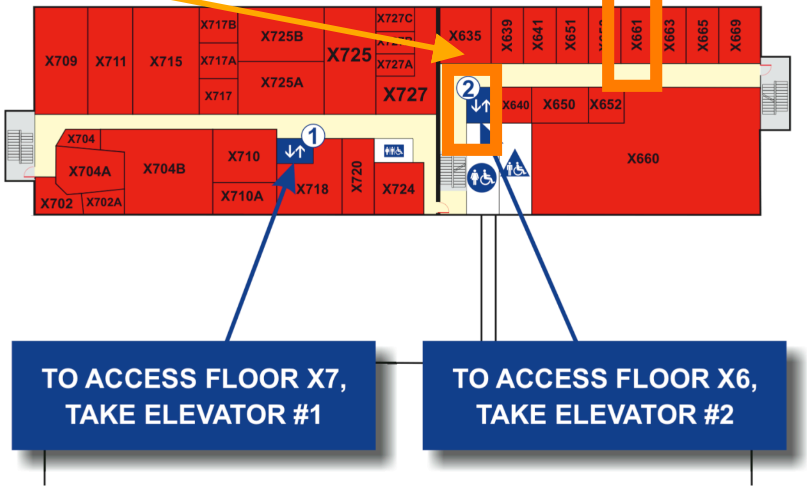


Xwing entrances facing Dempster

this elevator to X6

me!





TO ACCESS FLOOR X7, TAKE ELEVATOR #1

TO ACCESS FLOOR X6, TAKE ELEVATOR #2

Followup

- Q: identifiers - what about “.”?
  - `System.out.println(“hey, what’s the story?”);`
- A: not allowed in simple identifiers
  - qualified identifiers: sequence of simple identifiers, separated by “.”
  - stay tuned for more on scope, namespace and packages

Reading This Week

- Rest of Chap 2
  - 2.3-4, 2.6-2.10
- Rest of Chap 4
  - 4.3-4.7

Recap: Declaration and Assignment

- Variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- Variable name is specified by identifier
- Syntax:
  - *typeName variableName;*
  - *typeName variableName = value;*
    - can declare and assign in one step

- Java first computes value on right side
- Then assigns value to variable given on left side

```
x = 4 + 7;
```

Recap: Assignment Statements

- Here’s an occasional point of confusion:

```
a = 7;         // what’s in a?
b = a;         // what’s in b?
               // what’s in a now???
System.out.println(“a is “ + a + “b is “ +b);
a = 8;
System.out.println(“a is “ + a + “b is “ +b);
```

- Draw and fill in boxes for your variables at each time step if you’re confused

Recap: Expressions

- expression is combination of
  - one or more operators and operands
  - operator examples: +, *, /, ...
  - operand examples: numbers, variables, ...
- precedence: multiply/divide higher than add/subtract

Recap: Converting Between Types

- Doubles can simply be assigned ints
  - `double socks = 1;`
  - ints are subset of doubles
- Casting: convert from one type to another with information loss
- Converting from real to integer
  - `int shoes = (int) 1.5;`
- Truncation: fractional part thrown away
  - `int shoes = (int) 1.75;`
- Rounding: must be done explicitly
  - `shoes = Math.round(1.99);`

Recap: Primitive Data Types: Numbers

Type	Size	Min	Max
`byte`	1 byte	-128	127
`short`	2 bytes	-32,768	32,767
`int`	4 bytes	-2,147,483,648	2,147,483,647
`long`	8 bytes	**-9,223,372,036,854,775,808**	**9,223,372,036,854,775,807**
`float`	4 bytes	**approx -3.4E38 (7 sig.digits)**	**approx 3.4E38 (7 sig.digits)**
`double`	8 bytes	**approx -1.7E308 (15 sig. digits)**	**approx 1.7E308 (15 sig. digits)**

- Primary primitives are **`int`** and **`double`**
  - three other integer types
  - one other real type

Recap: Primitives: Non-numeric

- Character type
    - named char
    - Java uses the Unicode character set so each char occupies 2 bytes of memory.
- Boolean type
    - named boolean
    - variables of type boolean have only two valid values
        - true and false
    - often represents whether particular condition is true
    - more generally represents any data that has two states
        - yes/no, on/off

Recap: Constants

- Things that do not vary
  - unlike variables
  - will never change
- Syntax:
  - final *typeName variableName;*
  - final *typeName variableName = value;*
- Constant names in all upper case
  - Java convention, not compiler/syntax requirement

Recap: Avoiding Magic Numbers

- magic numbers: numeric constants directly in code
  - almost always bad idea!
    - hard to understand code
    - hard to make changes
    - typos possible
  - use constants instead

Programming

- Programming is all about specifiying
  - data that is to be manipulated or acted upon
  - operations that can act upon data
  - order in which operations are applied to data

- So far: specify data using primitive data types
  - come with pre-defined operations like

    +, -, *, and /

Programming with Classes

- What if data we want to work with is more complex these few primitive data types?

Programming with Classes

- What if data we want to work with is more complex these few primitive data types?

- We can make our own data type: create a class
  - specifies nature of data we want to work with
  - operations that can be performed on that kind of data
- Operations defined within a class called methods

Programming with Classes

- Can have multiple variables of primitive types (int, double)
  - each has different name
  - each can have a different value

```
int x = 5;
int y = 17;
```

- Similar for classes: can have multiple instances of class String
  - each has different name
  - each can have different value

```
String name = “Tamara Munzner”;
String computerName = “pangolin”;
```

Programming with Objects

- Object: specific instance of a class

- Classes are templates for objects
  - programmers define classes
  - objects created from classes

Object Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
        firstname = new String (“Kermit");
        lastname = new String (“theFrog");
        System.out.println("I am not " + firstname
                           + " " + lastname);
    }
}
```

Object Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
        firstname = new String ("Kermit");
        lastname = new String (“theFrog");
        System.out.println("I am not " + firstname
                           + " " + lastname);
    }
}
```

- Declare two different String objects
  - one called **`firstname`** and one called **`lastname`**

Object Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
```

- Variable declaration does not create objects!

Object Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
```

- Variable declaration does not create objects!
  - just tells compiler to set aside spaces in memory with these names
- Spaces will not actually hold the whole objects
  - will hold references: pointers to or addresses of objects
  - objects themselves will be somewhere else in memory

Object Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
        firstname = new String ("Kermit");
        lastname = new String (“theFrog");
        System.out.println("I am not " + firstname
                                   + " " + lastname);
    }
}
```

- So `firstname` and `lastname` will not contain String objects
  - contain references to String objects

Constructors

- Constructor: method with same name as class
  - always used with **`new`**
  - actually creates object
  - typically initializes with data

```
firstname = new String (“Kermit");
```

Object Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
        firstname = new String (“Kermit");
        lastname = new String (“theFrog");
        System.out.println("I am not " + firstname
                                  + " " + lastname);
    }
}
```

- Now create new instance of the String class
  - String object with data “Kermit”
- Puts object somewhere in memory
  - puts address of the object’s location in **`firstname`**:
    `firstname` holds reference to String object with data “Kermit”

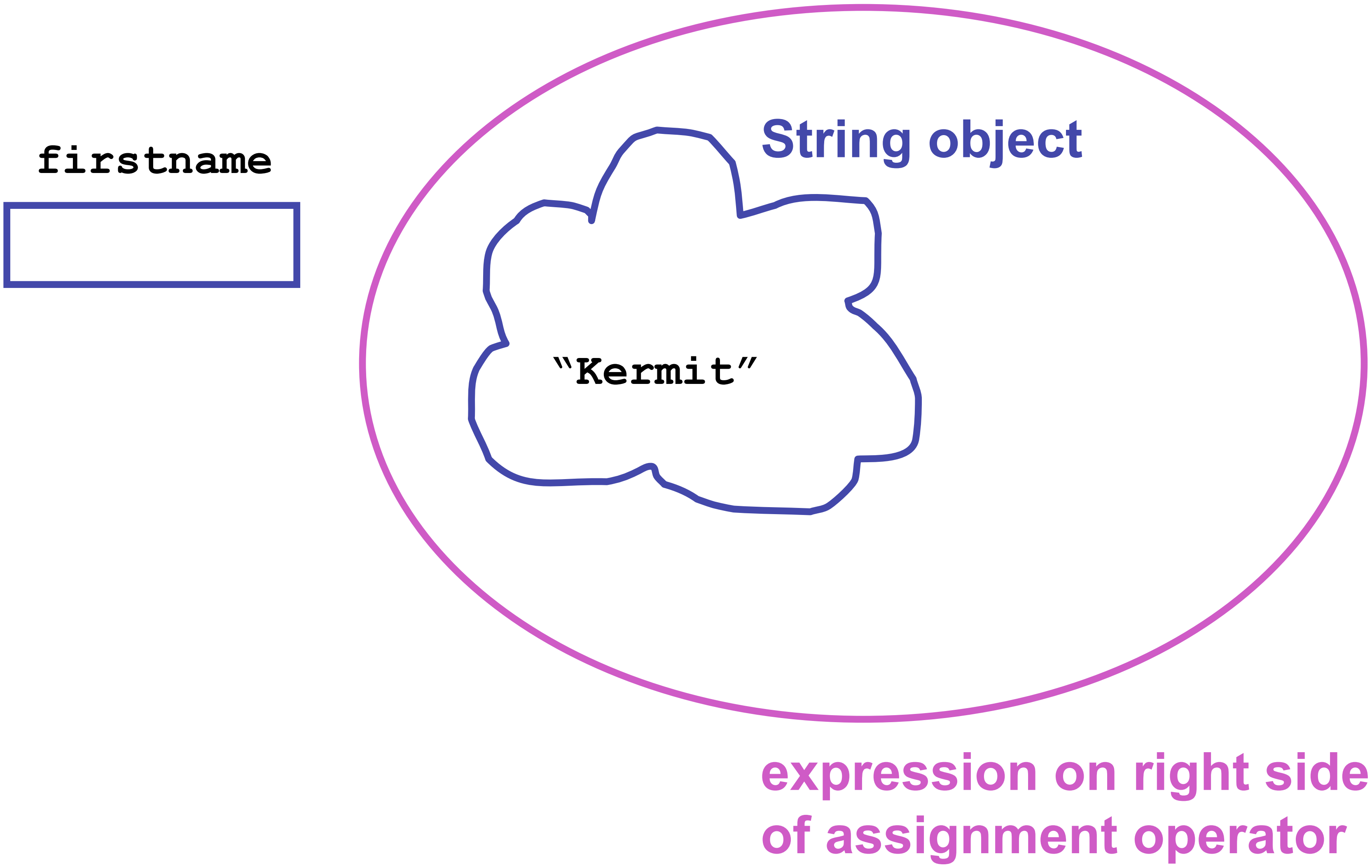
Object Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
        firstname = new String (“Kermit");
        lastname = new String (“theFrog");
        System.out.println("I am not " + firstname
                                   + " " + lastname);
    }
}
```

- New operator and String constructor method instantiate (create) new instance of String class (a new String object)

Object Example

Object Example



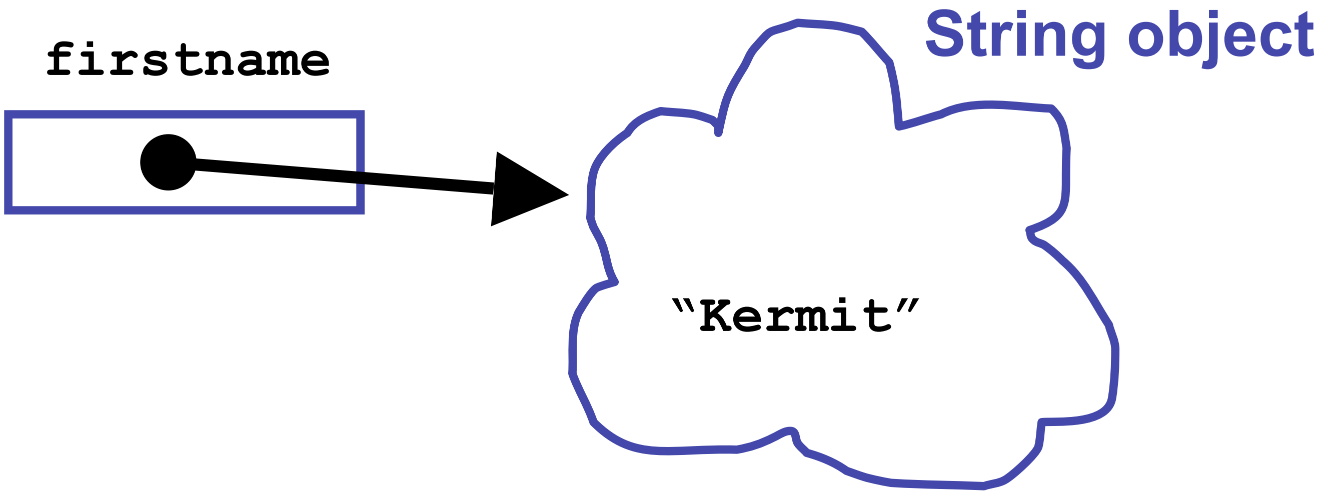
firstname

String object

"Kermit"

expression on right side of assignment operator

Object Example



firstname

String object

“Kermit”

bind variable to
expression on right side
of assignment operator

Object Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
        firstname = new String (“Kermit");
        lastname = new String (“theFrog");
        System.out.println("I am not " + firstname
                           + " " + lastname);
    }
}
```

- And so on

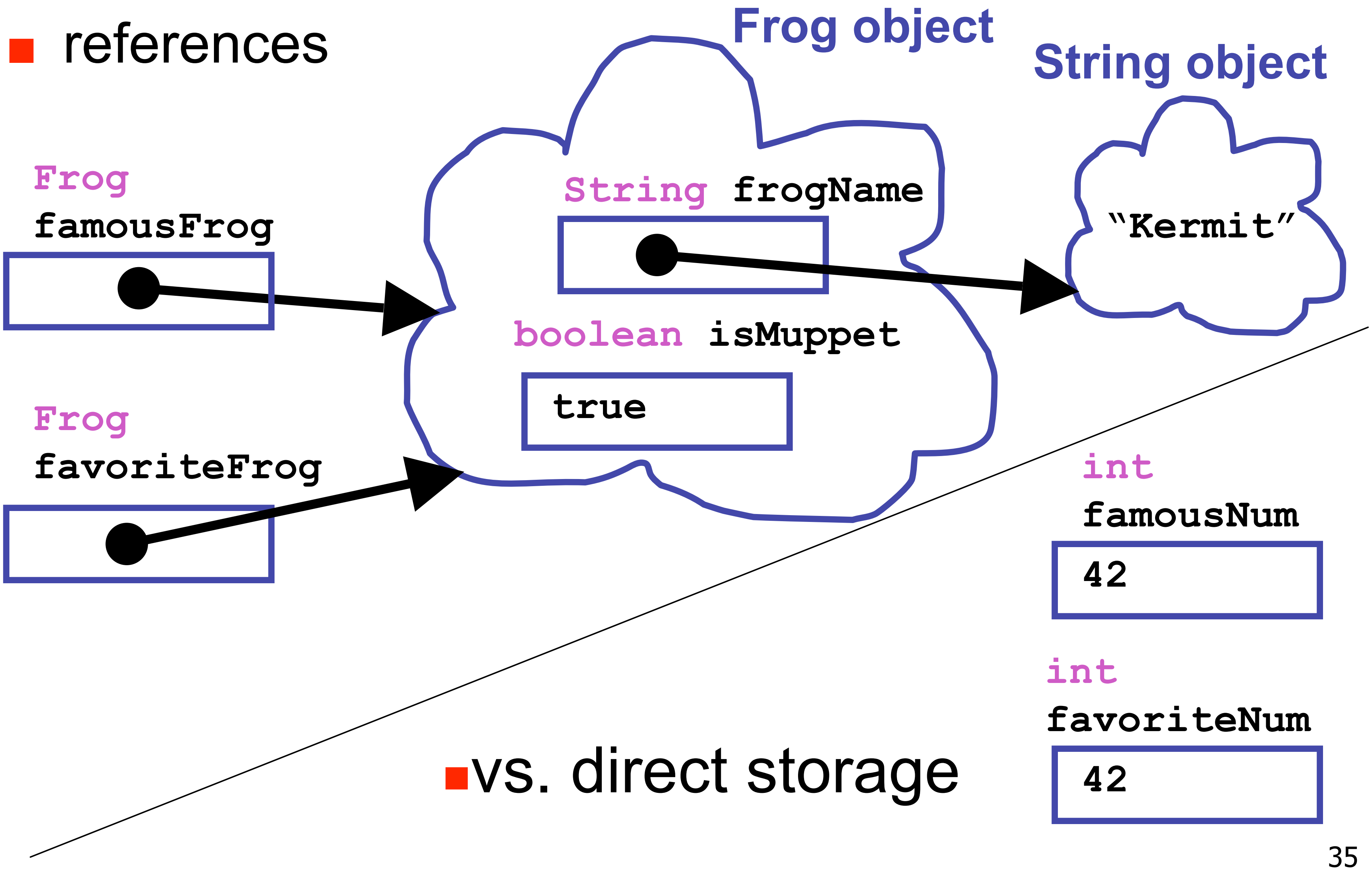
Object Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname = new String (“Kermit");
        String lastname = new String (“theFrog");
        System.out.println("I am not " + firstname
                          + " " + lastname);
    }
}
```

- Can consolidate declaration, assignment
  - just like with primitive data types

Objects vs. Primitives

- references



Frog object

String object

Frog famousFrog

Frog favoriteFrog

String frogName

boolean isMuppet

true

"Kermit"

int famousNum

42

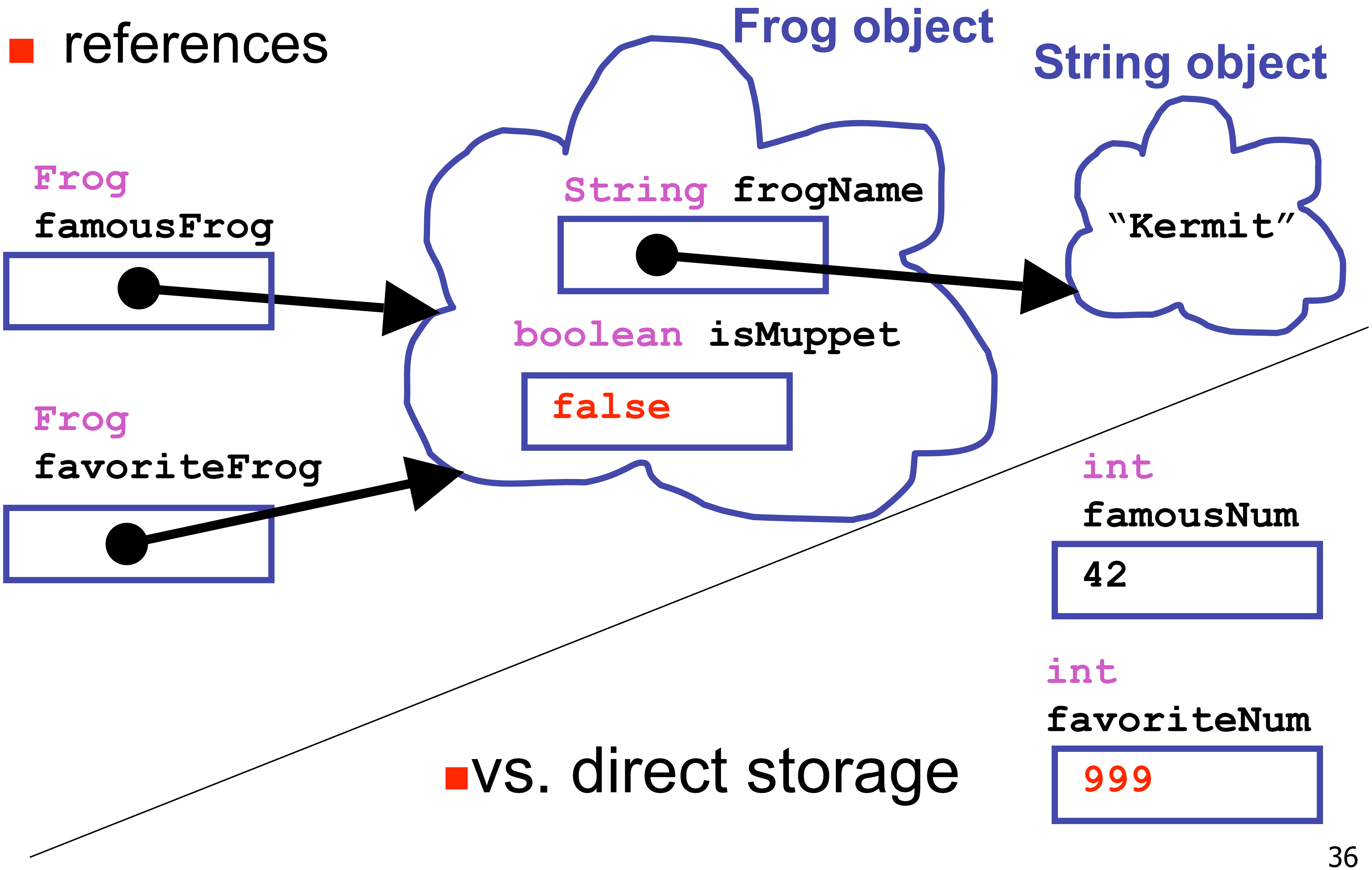
int favoriteNum

42

- vs. direct storage

Objects vs. Primitives

- references



- vs. direct storage

Class Libraries

- Before making new class yourself, check to see if someone else did it already
  - libraries written by other programmers
  - many built into Java
- Example
  - Java has single-character primitive data type
  - what if want to work with sequence of characters
  - String class already exists

API Documentation

- Online Java library documentation at http://java.sun.com/javase/6/docs/api/
  - textbook alone is only part of the story
  - let’s take a look!

- Everything we need to know: critical details
  - and often many things far beyond current need

- Classes in libraries are often referred to as Application Programming Interfaces
  - or just API

Some Available String Methods

`public String toUpperCase();`

Returns a new **`String`** object identical to this object but with all the characters converted to upper case.

`public int length();`

Returns the number of characters in this **`String`** object.

`public boolean equals( String otherString );`

Returns true if this **`String`** object is the same as **`otherString`** and false otherwise.

`public char charAt( int index );`

Returns the character at the given index. Note that the first character in the string is at index 0.

More String Methods

```
public String replace(char oldChar, char newChar);
```

Returns a new `String` object where all instances of `oldChar` have been changed into `newChar`.

```
public String substring(int beginIndex);
```

Returns new `String` object starting from `beginIndex` position

```
public String substring( int beginIndex, int endIndex );
```

Returns new `String` object starting from `beginIndex` position and ending at `endIndex` position

- up to but not including endIndex char:

```
substring(4, 7)     "o K"
```

H	e	l	l	o		K	e	r	m	i	t	F	r	o	g
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Questions?

String Method Example

```
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname = new String ("Kermit");
        String lastname = new String ("theFrog");
        firstname = firstname.toUpperCase();
        System.out.println("I am not " + firstname
                           + " " + lastname);
    }
}
```

- invoking methods
  - objectName.methodName();
  - remember (simple) identifiers can't have . in them

Methods and Parameters

- Class definition says what kinds of data and methods make up object
  - object is specific instance of class

```
String firstname = "Alphonse";
char thirdchar = firstname.charAt(2);
```

object

Methods and Parameters

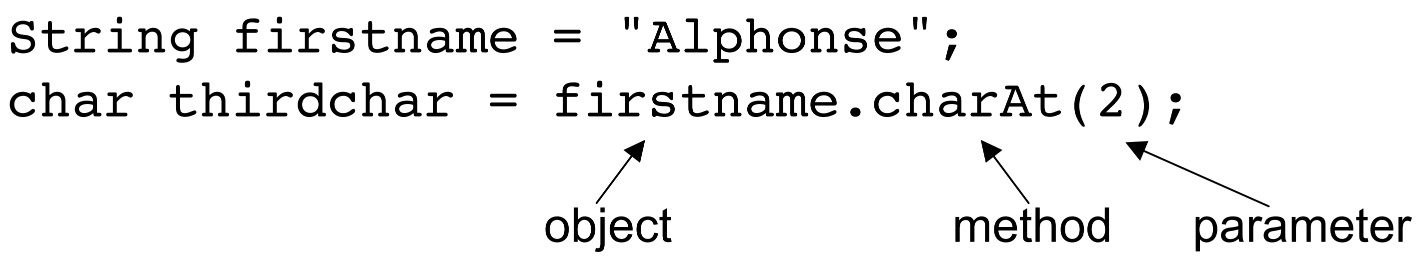
- Class definition says what kinds of data and methods make up object
  - object is specific instance of class
  - methods are how objects are manipulated

```
String firstname = "Alphonse";
char thirdchar = firstname.charAt(2);
```

object method

Methods and Parameters

- Class definition says what kinds of data and methods make up object
  - object is specific instance of class
  - methods are how objects are manipulated
  - pass information to methods with parameters
    - inputs to method call
    - tell `charAt` method which character in the String object we're interested in



```
String firstname = "Alphonse";
char thirdchar = firstname.charAt(2);
```

object

method

parameter

Parameters

- Methods can have multiple parameters
  - API specifies how many, and what type

```
public String replace(char oldChar, char newChar);

     String animal = "mole";
     animal.replace('m', 'v');

public String substring( int beginIndex, int endIndex );

     animal = "aardwolf";
     String newanimal = animal.substring(4,8);
     System.out.println(newanimal);              // wolf
```

Explicit vs. Implicit Parameters

- Explicit parameters given between parentheses
- Implicit parameter is object itself
- Example: **`substring`** method needs
  - **`beginIndex, endIndex`**
  - but also the string itself!

```
animal = "aardwolf";
System.out.println(animal);                    // aardwolf
String newanimal = animal.substring(4,8);
System.out.println(newanimal);                 // wolf
```

- All methods have single implicit parameters
  - can have any number of explicit parameters
    - none, one, two, many…

Parameters

- Most of the time we'll just say parameters, meaning the explicit ones

Return Values

- Methods can have return values
- Example: `charAt` method result
  - return value, the character 'n', is stored in `thirdchar`

```
String firstname = "kangaroo";
char thirdchar = firstname.charAt(2);
  return value     object    method parameter
```

Return Values

- Methods can have return values
- Example: `charAt` method result
  - return value, the character 'n', is stored in `thirdchar`

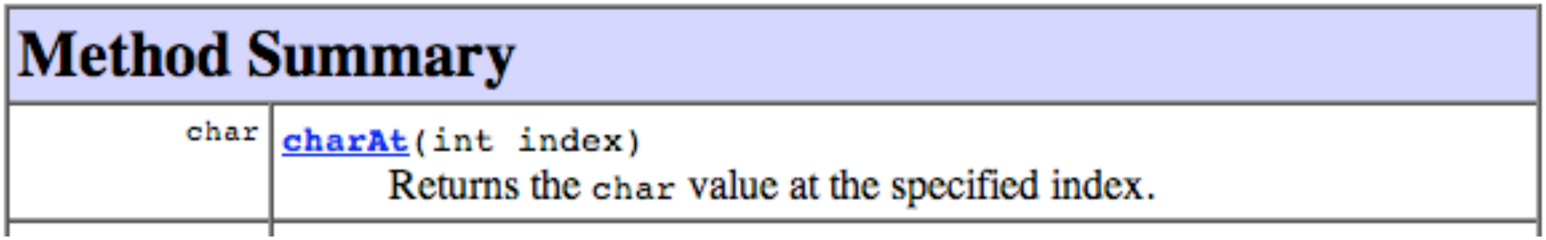
```
String firstname = "kangaroo";
char thirdchar = firstname.charAt(2);
  return value     object    method parameter
```

- Not all methods have return values
- Example: `println` method does not return anything
  - prints character 'n' on the monitor, but does not return that value
  - printing value and returning it are not the same thing!

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

Return Values

- Again, API docs tell you
  - how many explicit parameters
  - whether method has return value
  - what return value is, if so

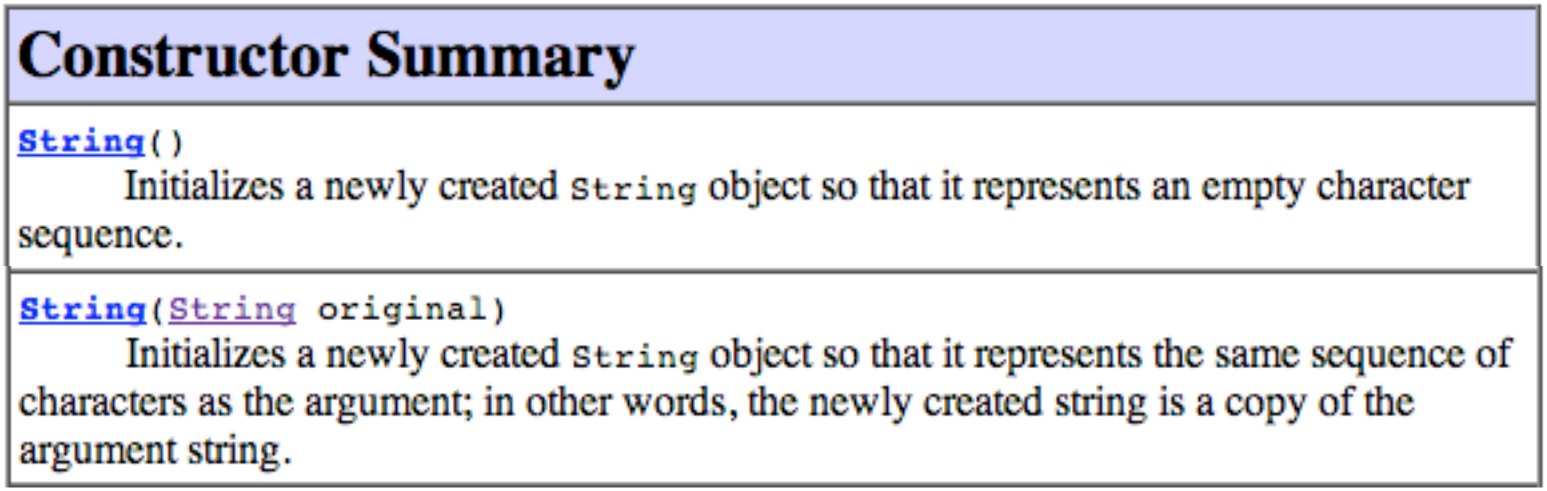


Method Summary	
`char`	`charAt(int index)` Returns the `char` value at the specified index.

- No return value indicated as `void`

Constructors and Parameters

- Many classes have more than one constructor, taking different parameters
  - use API docs to pick which one to use based on what initial data you have



Constructor Summary
`String()` Initializes a newly created `String` object so that it represents an empty character sequence.
`String(String original)` Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

```
a
animal = new String("kangaroo");
```

Accessors and Mutators

- Method that only retrieves data is accessor
  - read-only access to the value
  - example: charAt method of String class
- Method that changes data values internally is mutator
  - Stay tuned for examples of mutators, we haven't seen any yet
  - String class has no mutator methods
- Accessor often called getters
- Mutators often called setters
  - names often begin with get and set, as in getWhatever and setWhatever