



# Java Fundamentals

Nam Vu  
Principle Software Engineer  
Dec 2014



Use pptx extension instead of ppt extension when saving a file from now on.

## Agenda

There are two Agendas in this presentation. One in slide 2, the other in 5

- |     |               |    |
|-----|---------------|----|
| I.  | Java Basics   | xx |
| II. | Java Advanced | xx |

There is no course objectives in this presentation. Please apply Instructional Design Template to get the right structure

# Audience and Prerequisite

- The course is for any one who wants to learn Java
- The following are beneficial if you already have knowledge and experiences as:
  - Created and compiled programs with C/C++/Java

# Assessment Disciplines

- Class Participation: at least 80% of course time
- Assignment: get at least 70/100 score for final exercise

# Agenda

- Java Basics
  - Basic syntax
  - Operators and flow control
  - Primitive types and wrapper classes
- Java Advanced
  - Collection Framework
  - Exception Handling
  - Basic I/O
  - Serialization
  - Thread and Concurrency
  - Networking: Socket



The title doesn't match with the item in Agenda, which is Java Basics

## Java Basics and OOP

# Overview

- Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).
- Java is:
  - Object Oriented
  - Platform independent
  - Simple
  - Multithreaded
  - Distributed

# Basic Syntax

- Classes
- Methods
- Constructors
- Instance variables
- Local variables
- Class variables
- Objects



# Basic Syntax

- Identifiers:

- All identifiers should begin with a letter (A to Z or a to z), currency character (\$) or an underscore (\_).
- After the first character identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly identifiers are case sensitive.
- Examples of legal identifiers: `age`, `$salary`, `_value`, `__1_value`
- Examples of illegal identifiers: `123abc`, `-salary`

- Modifiers

- Access Modifiers: `default`, `public`, `protected`, `private`
- Non-access Modifiers: `static`, `final`, `abstract`, `synchronized`

# Demo 1: Greeting

- Points to remember:
  - Java coding convention
  - Review: class, constructors, methods, variables, constants ...



# Basic Operators

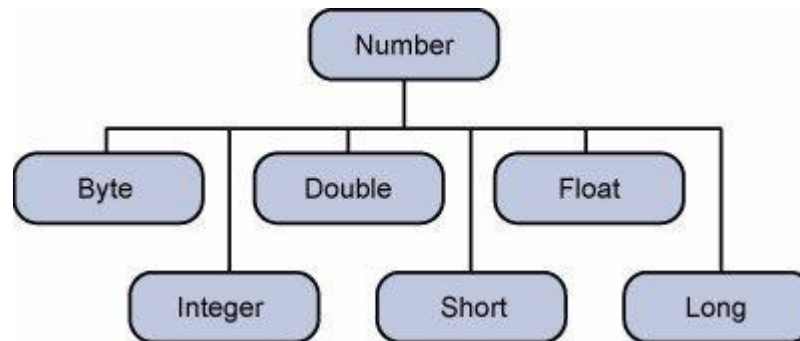
- Arithmetic Operators: +, -, \*, /, %, ++, --
- Relational Operators: ==, !=, >, <, >=, <=
- Bitwise Operators: &, |, ^...
- Logical Operators: &&, ||, !
- Assignment Operators: =, +=, -=, \*=, /=, %=, <<=, >>=, &=, ^=, |=
- Misc Operators:
  - Conditional operator (?:)  
variable x = (expression) ? value if true : value if false
  - Instance of  
( Object reference variable ) instanceof (class/interface type)

# Flow Control

- Loops:
  - while Loop
  - do...while Loop
  - for Loop
- Decision making:
  - if ... else if ....else
  - switch
- continue, break:

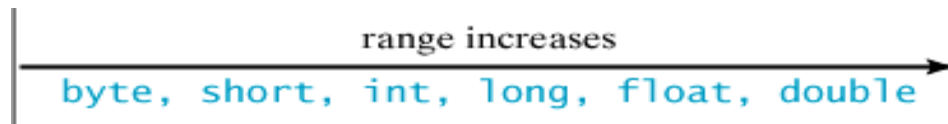
# Primitive Types and Wrapper Classes

- Logical – boolean
- Textual – char
- Integral – byte, short, int, and long
- Floating – double and float
- Wrapping Classes:



# Primitive Types and Wrapper Classes

<i>Name</i>	<i>Range</i>	<i>Storage Size</i>
byte	$-2^7$ (-128) to $2^7 - 1$ (127)	8-bit signed
short	$-2^{15}$ (-32768) to $2^{15} - 1$ (32767)	16-bit signed
int	$-2^{31}$ (-2147483648) to $2^{31} - 1$ (2147483647)	32-bit signed
long	$-2^{63}$ to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
float	Negative range: $-3.4028235E + 38$ to $-1.4E-45$ Positive range: $1.4E-45$ to $3.4028235E + 38$	32-bit IEEE 754
double	Negative range: $-1.7976931348623157E+308$ to $-4.9E-324$ Positive range: $4.9E-324$ to $1.7976931348623157E+308$	64-bit IEEE 754



# Primitive Types and Wrapper Classes

Variable	Value
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0
char	"
boolean	false
All reference types	null

# String

- String: String greeting = "Hello world!";
- StringBuilder
- StringBuffer

```
StringBuffer s1 = new StringBuffer("Java");  
StringBuffer s2 = new StringBuffer("HTML");
```

Show the results of the following expressions of `s1` after each statement. Assume that the expressions are independent.

(1) s1.append(" is fun");	(7) s1.deleteCharAt(3);
(2) s1.append(s2);	(8) s1.delete(1, 3);
(3) s1.insert(2, "is fun");	(9) s1.reverse();
(4) s1.insert(1, s2);	(10) s1.replace(1, 3, "Computer");
(5) s1.charAt(2);	(11) s1.substring(1, 3);
(6) s1.length();	(12) s1.substring(2);





# Java Advance

## Collection Framework

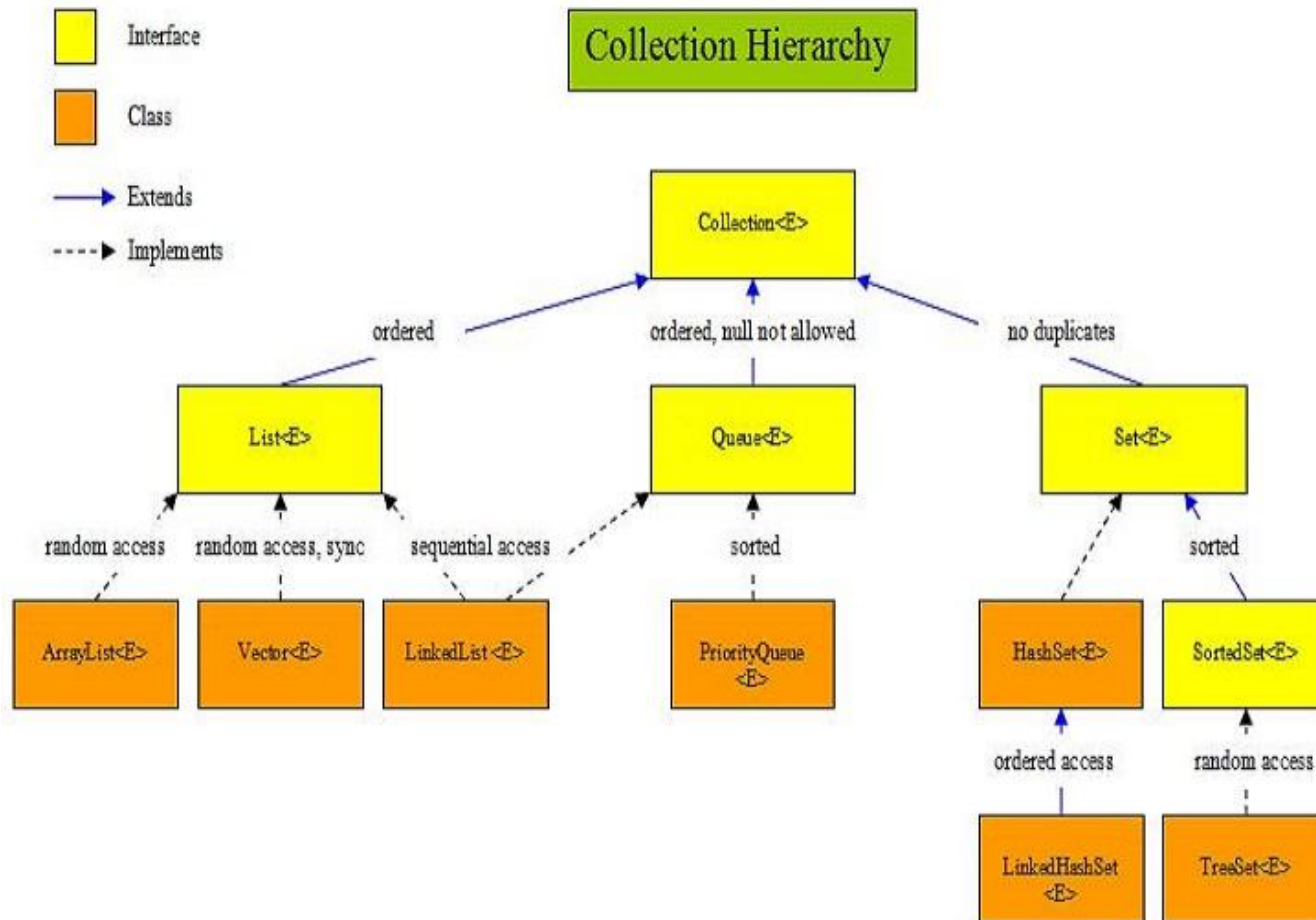
# Java Collections Framework

- A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:
  - Interfaces: Collection, List, Map
  - Implementations, i.e., Classes: ArrayList, HashSet, HashMap
  - Algorithms: These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces

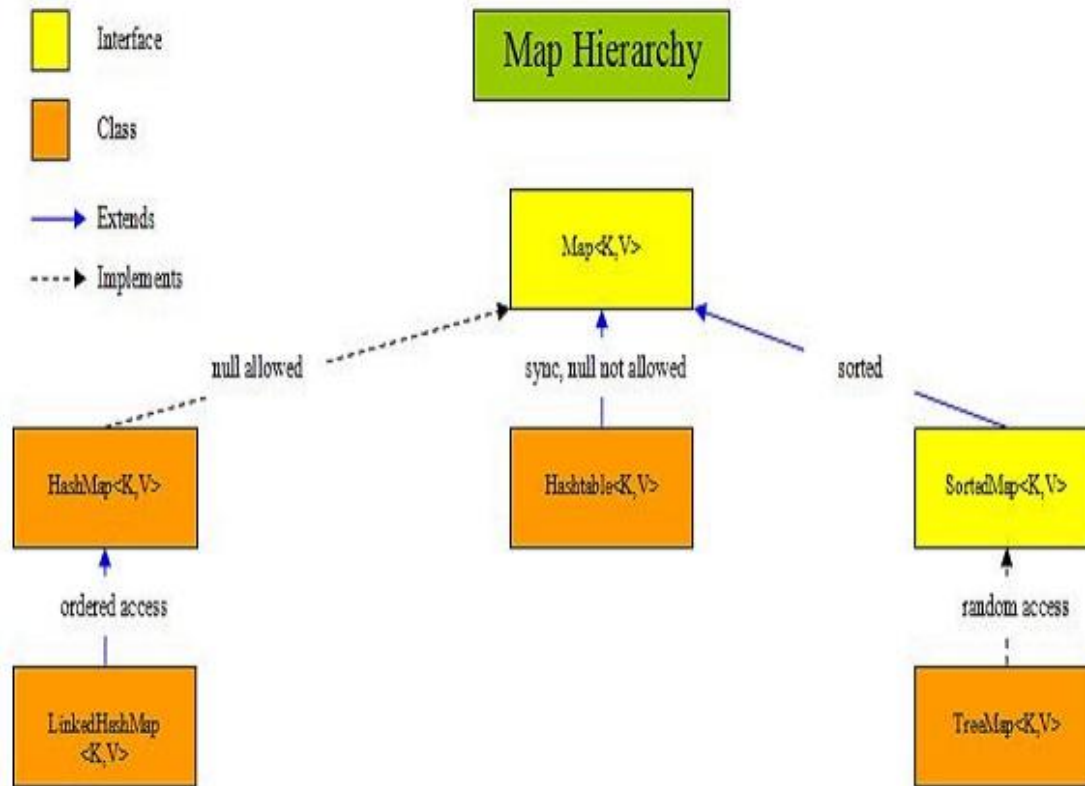
# Java Collections Framework

SN	Interfaces with Description
1	<b>The Collection Interface</b> This enables you to work with groups of objects; it is at the top of the collections hierarchy.
2	<b>The List Interface</b> This extends <b>Collection</b> and an instance of List stores an ordered collection of elements.
3	<b>The Set</b> This extends Collection to handle sets, which must contain unique elements
4	<b>The SortedSet</b> This extends Set to handle sorted sets
5	<b>The Map</b> This maps unique keys to values.
6	<b>The Map.Entry</b> This describes an element (a key/value pair) in a map. This is an inner class of Map.
7	<b>The SortedMap</b> This extends Map so that the keys are maintained in ascending order.
8	<b>The Enumeration</b> This is legacy interface and defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects. This legacy interface has been superceded by Iterator.

# Java Collections Framework



# Java Collections Framework



# Java Collections Framework

- Iterator: In general, to use an iterator to cycle through the contents of a collection, follow these steps:
  - Obtain an iterator to the start of the collection by calling the collection's `iterator()` method.
  - Set up a loop that makes a call to `hasNext( )`. Have the loop iterate as long as `hasNext()` returns true.
  - Within the loop, obtain each element by calling `next()`.

# Java Collections Framework

- The Methods Declared by Iterator:

SN	Methods with Description
1	<b>boolean hasNext( )</b> Returns true if there are more elements. Otherwise, returns false.
2	<b>Object next( )</b> Returns the next element. Throws NoSuchElementException if there is not a next element.
3	<b>void remove( )</b> Removes the current element. Throws IllegalStateException if an attempt is made to call remove ( ) that is not preceded by a call to next( ).

No image description

# Java Collections Framework

- The Methods Declared by ListIterator:

SN	Methods with Description
1	<b>void add(Object obj)</b> Inserts obj into the list in front of the element that will be returned by the next call to next( ).
2	<b>boolean hasNext( )</b> Returns true if there is a next element. Otherwise, returns false.
3	<b>boolean hasPrevious( )</b> Returns true if there is a previous element. Otherwise, returns false.
4	<b>Object next( )</b> Returns the next element. A NoSuchElementException is thrown if there is not a next element.
5	<b>int nextIndex( )</b> Returns the index of the next element. If there is not a next element, returns the size of the list.
6	<b>Object previous( )</b> Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.
7	<b>int previousIndex( )</b> Returns the index of the previous element. If there is not a previous element, returns -1.
8	<b>void remove( )</b> Removes the current element from the list. An IllegalStateException is thrown if remove( ) is called before next( ) or previous( ) is invoked.
9	<b>void set(Object obj)</b> Assigns obj to the current element. This is the element last returned by a call to either next( ) or previous( ).

No image description



# Java Collections Framework

Comparator	Comparable
A comparator object is capable of comparing two different objects. The class is not comparing its instances, but some other class's instances. This comparator class must implement the <code>java.util.Comparator</code> interface	A comparable object is capable of comparing itself with another object. The class itself must implements the <code>java.lang.Comparable</code> interface in order to be able to compare its instances.
<code>int compare(Object o1, Object o2)</code>	<code>int compareTo(Object o1)</code>
<code>TreeSet(Comparator)</code> <code>java.util.Collections.sort(List, Comparator)</code>	<code>TreeSet()</code> <code>java.util.Collections.sort(List)</code>

No image description

## Demo 3: Collection Framework



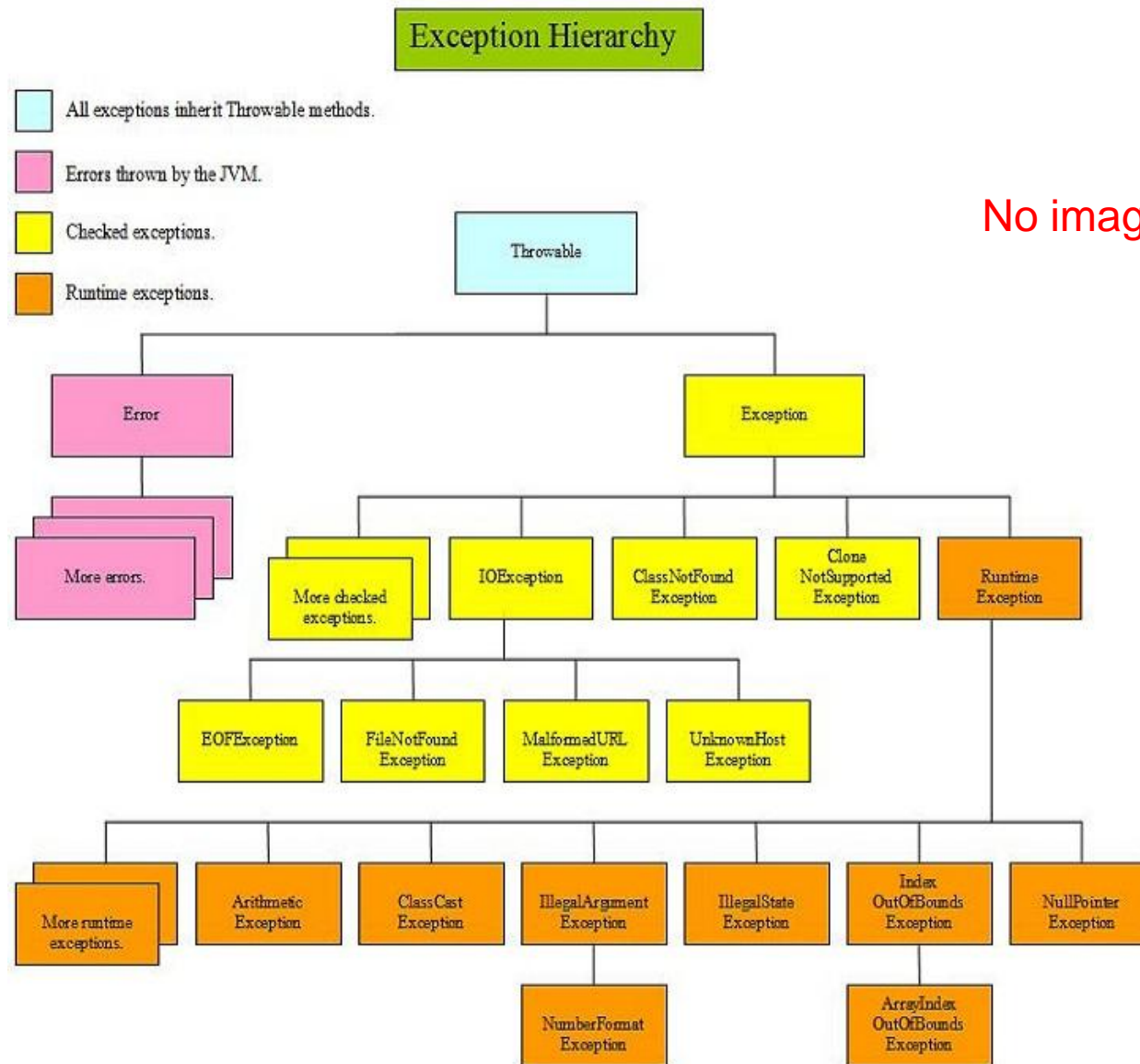
- Exercise:
  - Add Customer class which has: name, memberType (VIP, MEMBER, OTHERS) and list of orders
  - Sort orders in demo 2 by date, desc.
  - Display orders in console
- Points to remember:
  - Initialize collections
  - Manage collections
  - Order collections
  - Generic, advanced loop



# Java Advance

## Exceptions handling

# Exception Hierarchy



No image description



# Java Advance

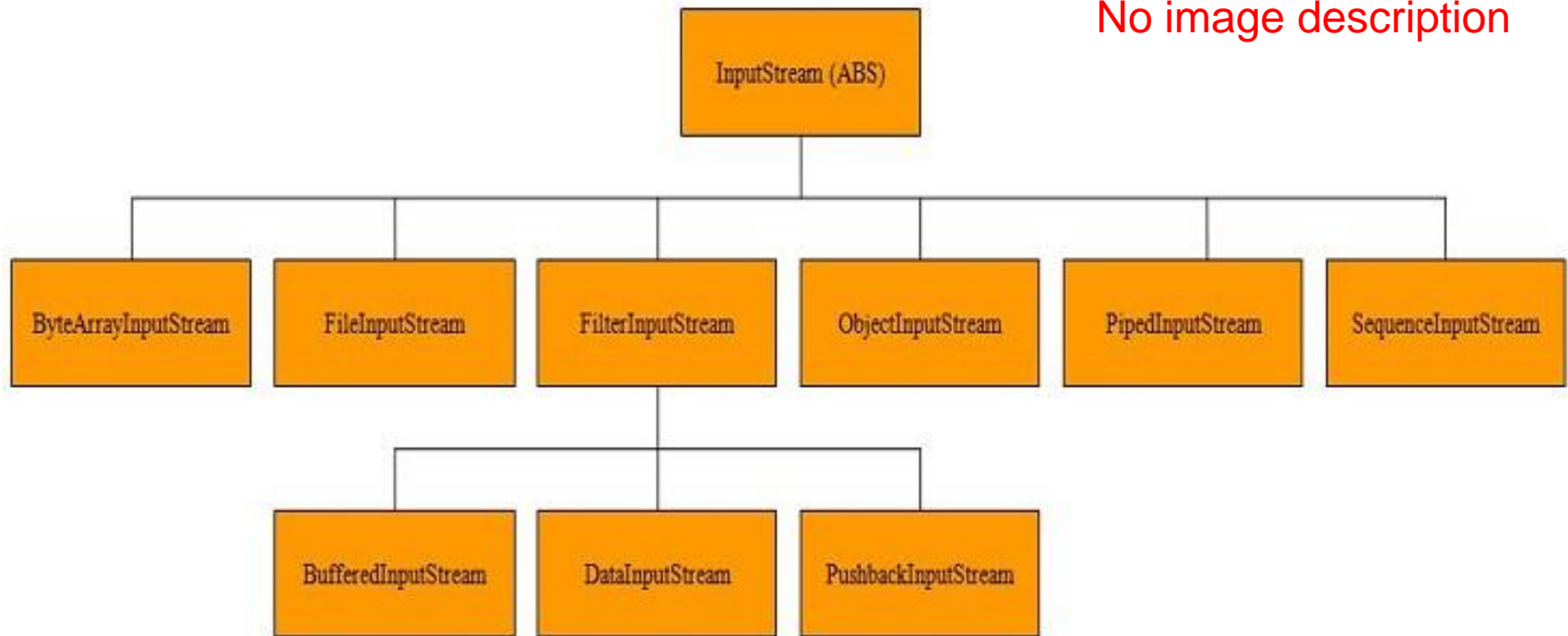
Basic I/O

# Basic I/O

- A *stream* can be thought of as a flow of data from a source or to a sink.
- A *source* stream initiates the flow of data, also called an input stream.
- A *sink* stream terminates the flow of data, also called an output stream.
- Sources and sinks are both *node streams*.
- Types of node streams:
  - Byte streams
  - Character streams
  - Buffered streams
  - Data streams
  - Object streams

# Basic I/O

## Byte Input Stream Hierarchy

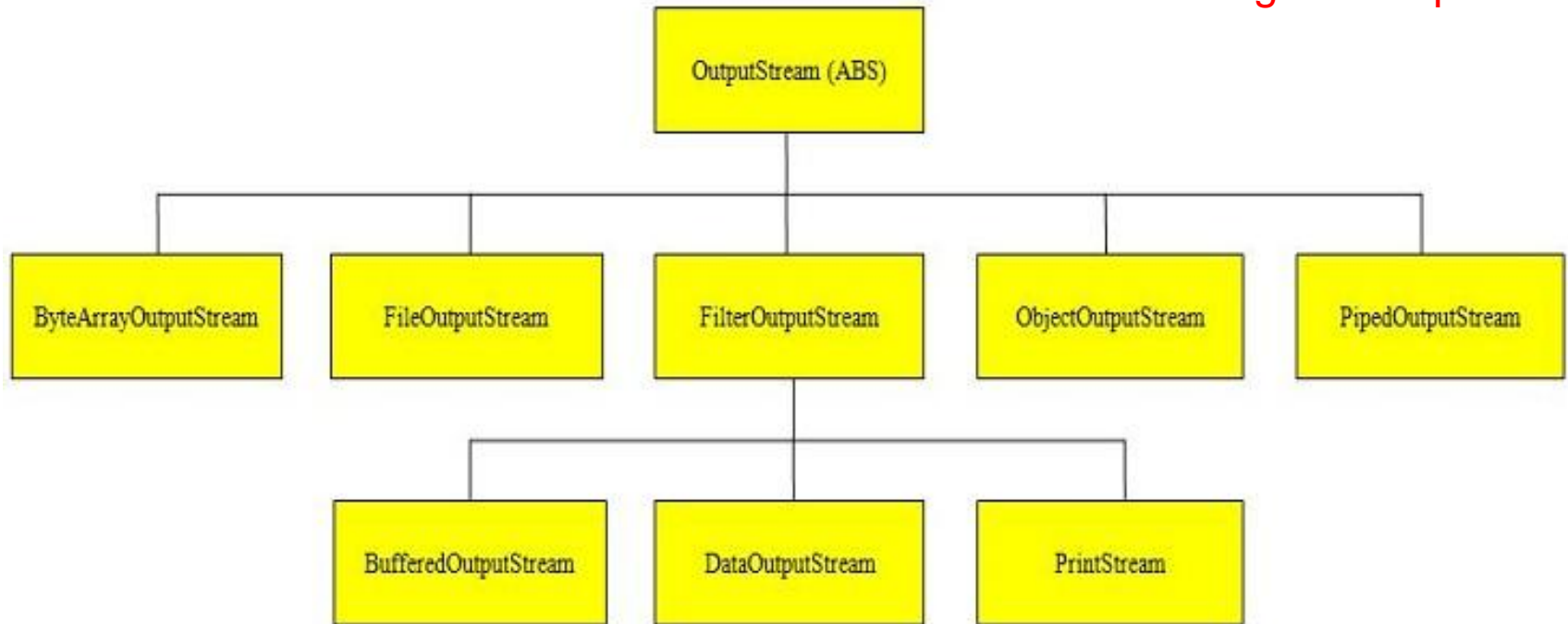




# Basic I/O

## Byte Output Stream Hierarchy

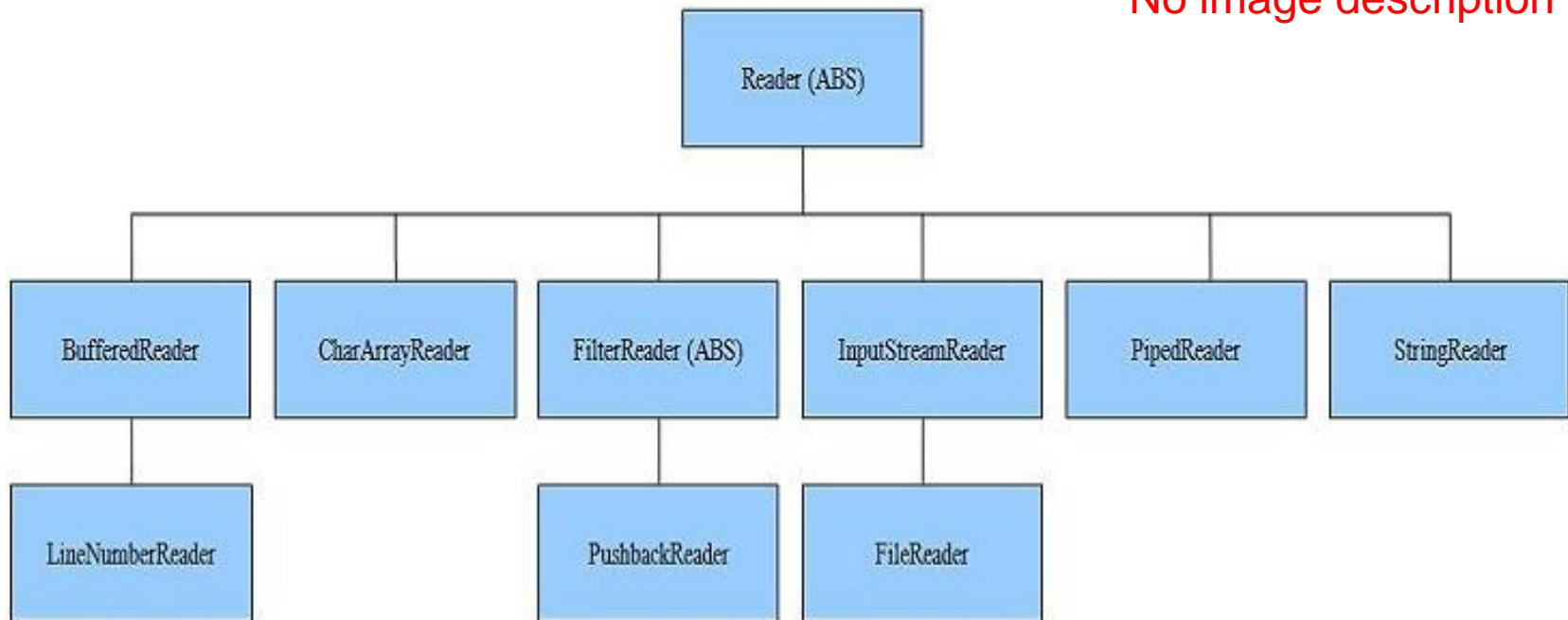
No image description





# Basic I/O

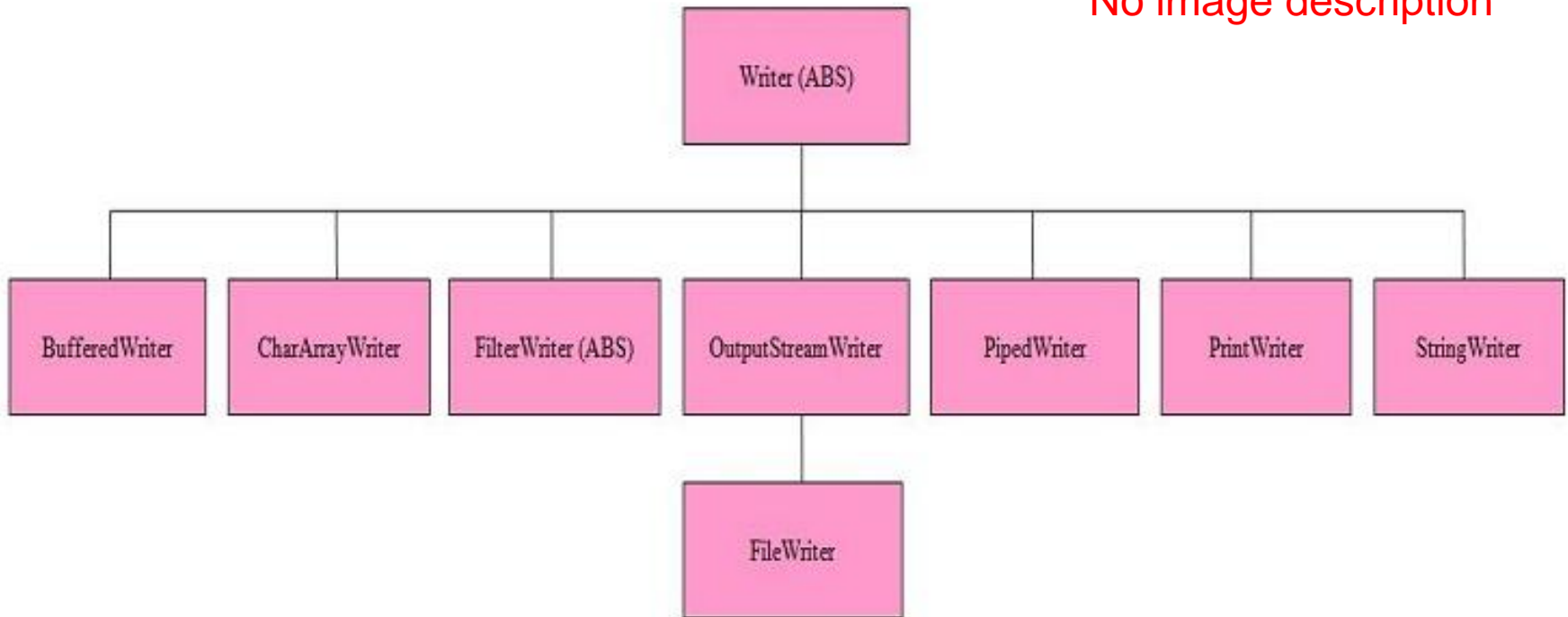
## Character Input Stream Hierarchy



# Basic I/O

## Character Output Stream Hierarchy

No image description



## Demo 4: Basic I/O



- Exercise: Save customer with orders to file
- Points to remember:
  - Use classes provided in I/O package to read and write from/to a file
  - Handle exception



# Java Advance

## Serialization

# Serialization

- Interface Serializable:
  - A marker interface, when implementing it, it enables classes to serialize/deserialize their state
  - Use ObjectOutputStream/ObjectInputStream to write/read an object to/from a stream (or file)

# Demo 5: Serialization



- Exercise:
  - Serialize customer with orders to a file
  - Deserialize customer from the file, print out to console and compare
- Points to remember:
  - Usage of Serializable interface and Object Input/Output Stream

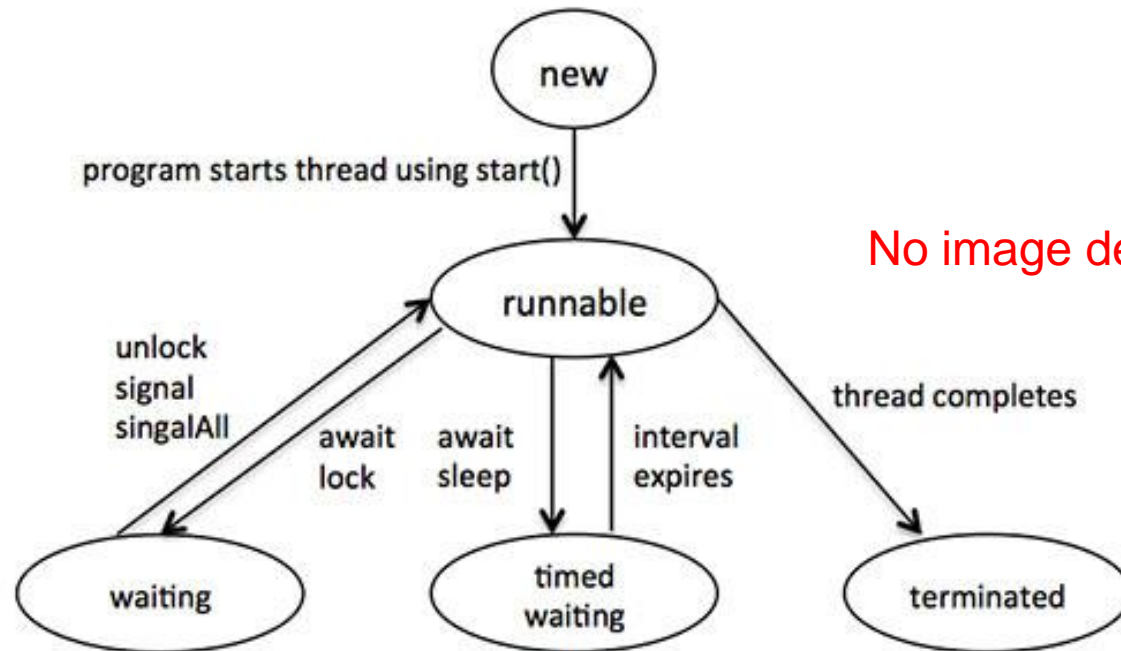


# Java Advance

## Multithreading

# Multithreading – Thread definition

- A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.
- States:



No image description



# Multithreading – Creating thread

- Two ways to create a thread:
  - Implement Runnable interface
  - Extend Thread class

# Multithreading - Concurrency

- Thread Interference
- Memory Consistency Errors
- Synchronization: synchronized methods, statements
- Deadlock
- Thread Communication: wait, notify, notifyAll

## Demo 6: Thread



- Exercise:
  - Create two threads accessing and modifying one customer.
  - Each thread prints the order to console.
  - Investigate the result
- Points to remember:
  - Create thread
  - Experience multithreading issues.



# Java Advance

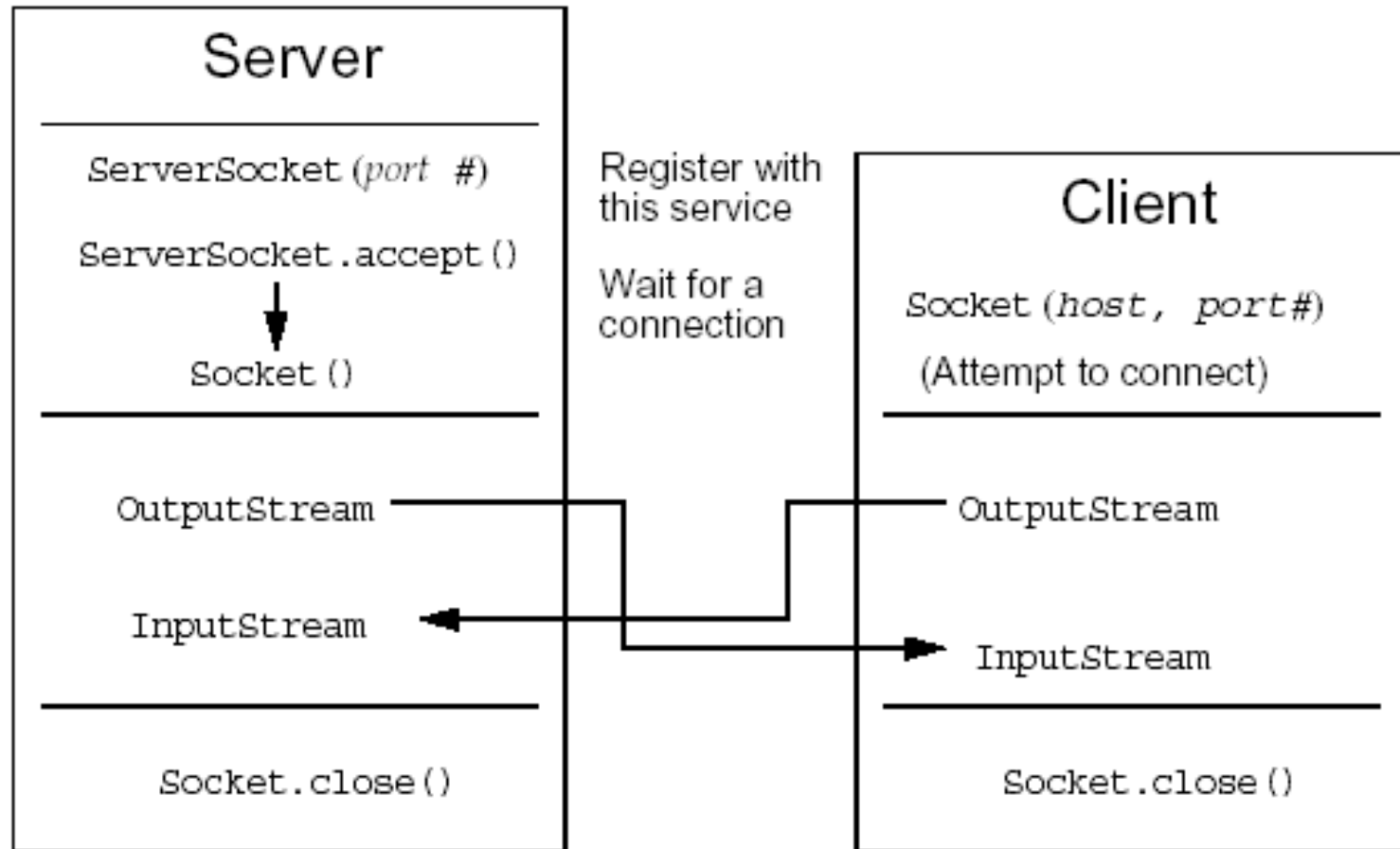
Sockets

# Sockets

- A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program
- The java.net package provides Socket that implements the client side of the connection
- The java.net package provides ServerSocket that implement the server side of the connection

# Sockets

No image description



# Sockets

- Open a socket.
- Open an input stream and output stream to the socket.
- Read from and write to the stream according to the server's protocol.
- Close the streams.
- Close the socket.

# Demo 7: Socket



- Exercise:
  - Create socket client and server.
  - Server hold an Order object.
  - When a client connecting, server sends client that order
  - Client shows order to console
- Points to remember:
  - Create socket
  - Communication between to sockets.





## Final Project

- This is a client – server application
- Client asks for a customer with a name
- If server doesn't have a customer with that name, it creates new one, otherwise it returns the existing one.
- Client can:
  - add order(s) to customer
  - print orders to console
  - send customer back to server to save
- Server saves (serializes) customer to file
- One customer has a list of orders, sorting by date (desc)
- Depending on customer type (VIP, MEMBER, OTHERS) customer receives different discount percentage



**Thank You**

Please add course  
revision history slide at the  
end.



BUSINESS SOLUTIONS  
TECHNOLOGY  
OUTSOURCING