Java

A platform independent programming language

Agenda

- Introduction
- Language Basics
- Arrays
- Classes and Objects
- Concept of OOPs
- Packages
- Exception Handling
- File Handling (IO Streams)

Agenda

- Multithreading
- Collection Framework
- Generics
- SQL
- JDBC
- Factory Pattern

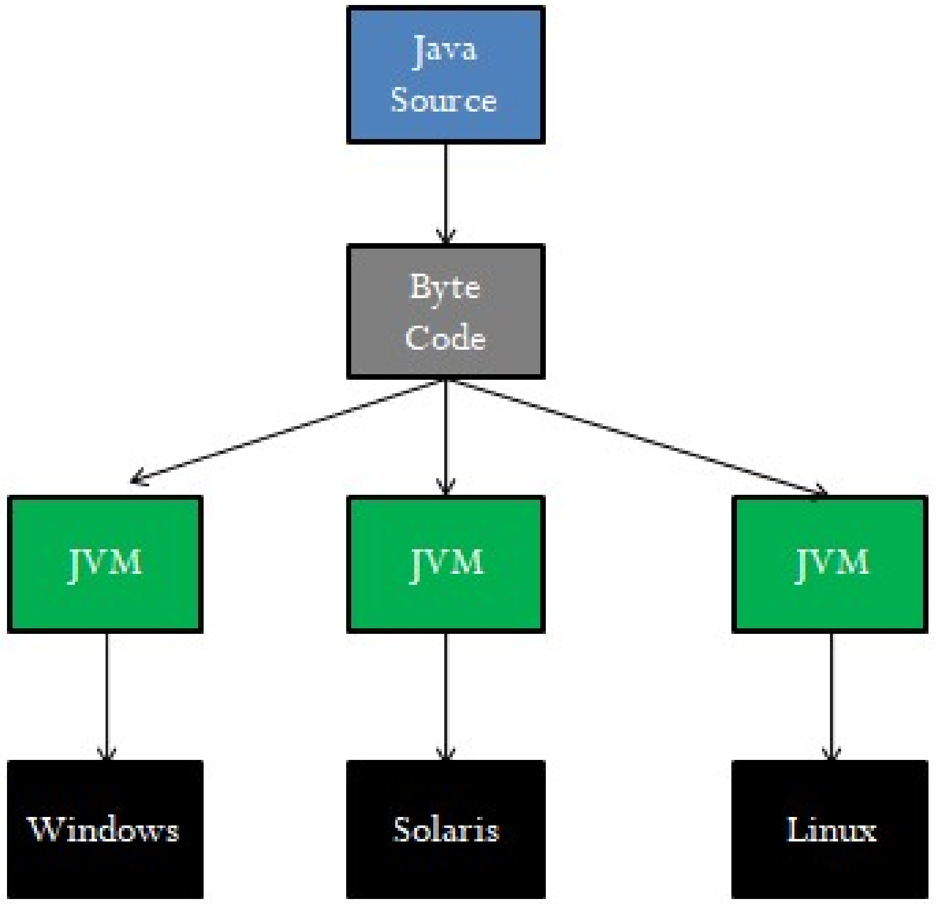
Hardware and Software

- JDK
- Editor or IDE (STS)
- Database (Derby)
- Windows / Mac / Linux OS
- 4 GB RAM

Introduction to Java

- Simple
- Platform Independent
- Object Oriented

Platform Independent



Object Oriented

- Object is a real world entity

- Objects will have
  - Properties
    - What object has
  - Behaviours
    - What object does

How to compile and run

- `javac` is the command used to compile
  - javac Filename.java

- `java` is the command used to run
  - java mainclass

JDK and JRE

- JDK stands for Java Development Kit
  - Development Environment

- JRE stands for Java Runtime Environment
  - Runtime Environment
  - Class Loader
  - Byte Code Verifier
  - JVM

Writing our first java program

```java
public class HelloApp
{
        public static void main(String[] args)
        {
                System.out.println("Welcome to Java World");
        }
}
```

```
Compiling
javac HelloApp.java

Running
java HelloApp

Output
Welcome to Java World
```

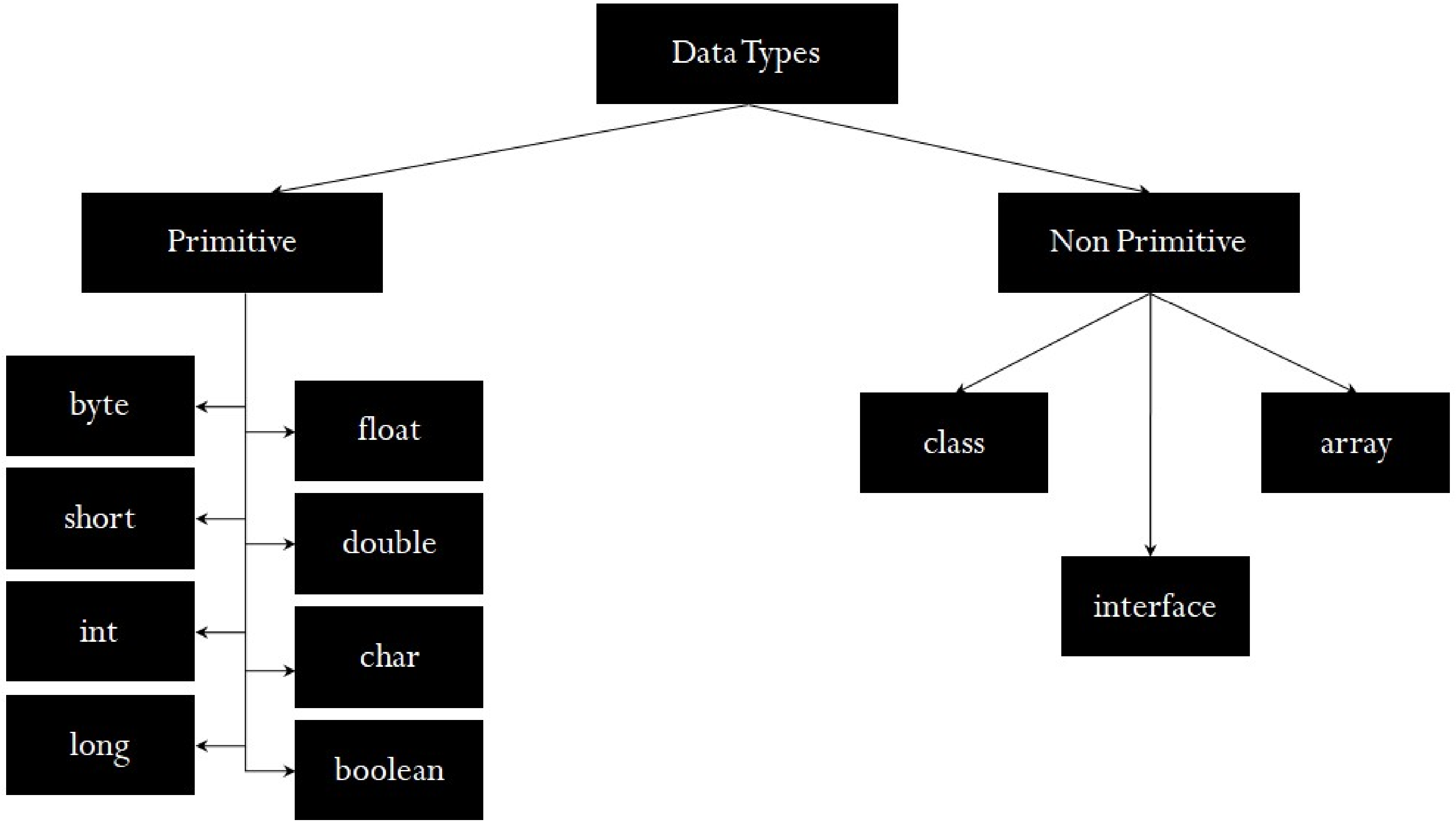
Java Fundamentals

Datatypes, Operators, Conditional Statements, Loops, Arrays

Fundamentals

- Datatypes
- Operators
- Conditional Statements
- Looping Constructs
- Arrays

Data Types


Data Types
Primitive
Non Primitive
byte
short
int
long
float
double
char
boolean
class
interface
array

Operators

- Types of Operators
  - Assignment
  - Arithmetic
  - Unary
  - Equality & Relational
  - Conditional

Operators (Contd.)

- Assignment

  =

- Arithmetic

  +, -, *, /, %

- Unary

  +, -, ++, --, !

- Equality & Relational

  ==, !=, <, >, <=, >=

- Conditional

  &&, ||

Conditional Statements

- You often want certain codes to be executed based on some conditions, then you can use the conditional statements like
  - if
  - if else
  - if else if … else ladder
  - switch

Looping Constructs

- When you want to repeatedly execute some codes you can use loops like
  - for loop
  - while loop
  - do while loop

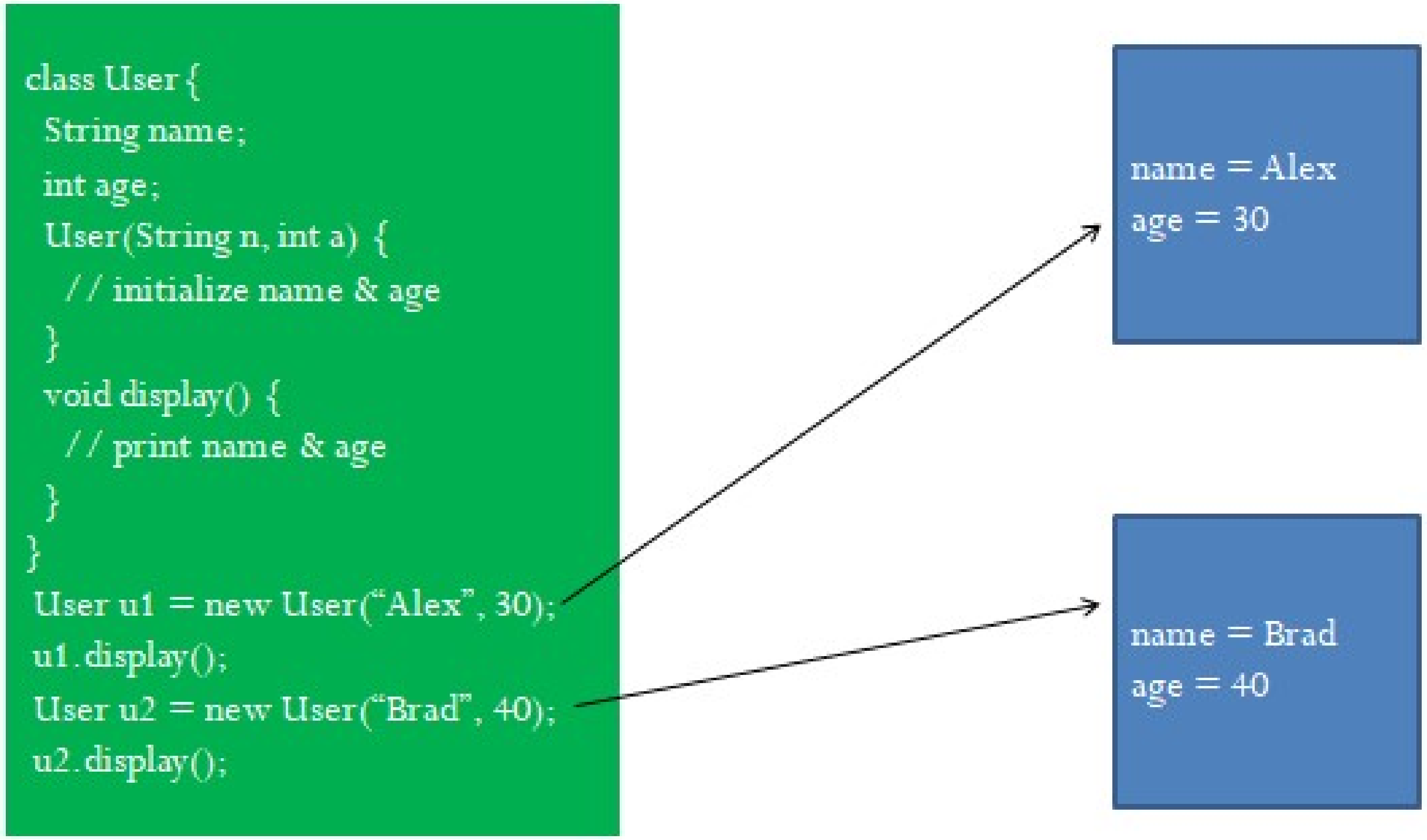
Classes & Objects

Constructors,Variables, Methods

Classes and Objects

- Class
  - Template of an object
  - Blueprint
- Object
  - Instance of a class,
  - Created with `new` keyword
  - Initialized through constructors

Classes and Objects



```
class User {
  String name;
  int age;
  User(String n, int a) {
    // initialize name & age
  }
  void display() {
    // print name & age
  }
}
User u1 = new User("Alex", 30);
u1.display();
User u2 = new User("Brad", 40);
u2.display();
```

Constructors

- Constructors
  - Called when objects are created
  - Name will be same as class name and wouldn’t have return type
  - Default constructor is created when you don’t provide any

Arrays

- Array
  - It is a container which stores fixed number of values of a single type
  - Declaring an array
    - Type[] arr = new Type[size];
  - You can create array for primitive & complex types

Object Arrays

- A single variable can have multiple objects through object array

- Suppose you want multiple user objects then you can use
  - *User[] users = new User[size];*

Passing values and objects

- Methods and Constructors can take parameters like
  - Primitive types and
  - Derived types

- Primitive types are pass by values

- Derived types are pass by reference

Object Oriented Features

Inheritance, Polymorphism, Abstraction & Encapsulation

OOPs Features

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

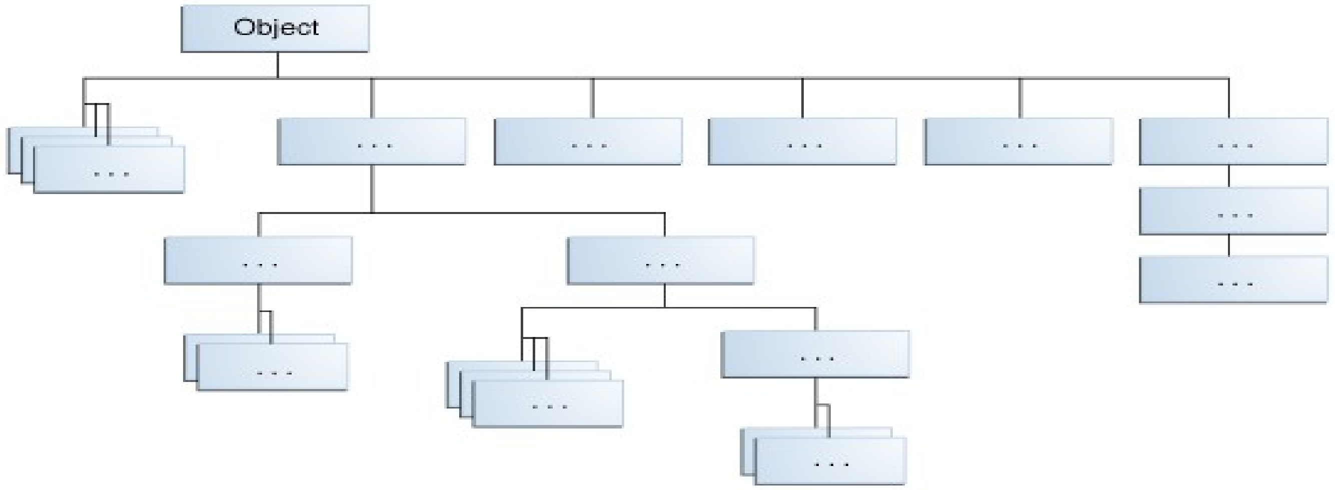
Inheritance

- Acquiring
  - Properties and
  - Behaviours

- It is achieved using `extends` keyword
  - Private members & constructors are not inherited to the subclass
  - `super` keyword is used to access super class members & call super class constructors

Object class

- Object class
  - Top most class
  - If a class doesn't extend any class then Object will be its super class



Types of Inheritance

Single level

$$\begin{tikzcd}
A \\
B \arrow[u]
\end{tikzcd}$$

Multi level

$$\begin{tikzcd}
A \\
B \arrow[u] \\
C \arrow[u]
\end{tikzcd}$$

Hierarchical

$$\begin{tikzcd}
& A & \\
B \arrow[ur] & C \arrow[u] & D \arrow[ul]
\end{tikzcd}$$

Casting Objects

- Suppose you have an hierarchy of classes like
  - Employee extends Person
  - Student extends Person

- You can create object of Student and assign to the Person variable, however the reverse is not always true
  - Person p = new Student(); // Auto Upcasting
  - Student s = (Student)p; // Explicit Downcasting

Constructor chain in Inheritance

- When classes are created with an inheritance hierarchy
  - Every subclass constructor implicitly calls its parent class default constructor

- super(args) can be used to call overloaded constructors
  - It must be always written on the very first line of the constructor

Polymorphism

- Poly means many and morphism means forms

- Polymorphism helps a method with the same name to perform multiple actions
  - Ex: A power button acts as both On/Off
  - Ex: A method can give different results based on the object you are using to call

Types of Polymorphism

- Compile Time Polymorphism
  - Achieved through method overloading
  - Done in the same class
- Runtime Polymorphism
  - Achieved through method overriding
  - Done in the subclasses

Encapsulation

- Encapsulation describes the ability of an object to hide its data and methods from the rest of the world.

- Private members
  – They can’t be accessed outside the class
  – Enclosing class must have public methods which access private members

Abstraction

- Abstraction means hiding the implementation details and showing only functionality to the user.

- It can be achieved in two ways
  - Abstract class
  - Interfaces

Abstract Classes and Methods

- An *abstract class* is a class that is declared
  - Abstract classes cannot be instantiated,
  - They can be subclassed.

- An *abstract method* is a method that is declared

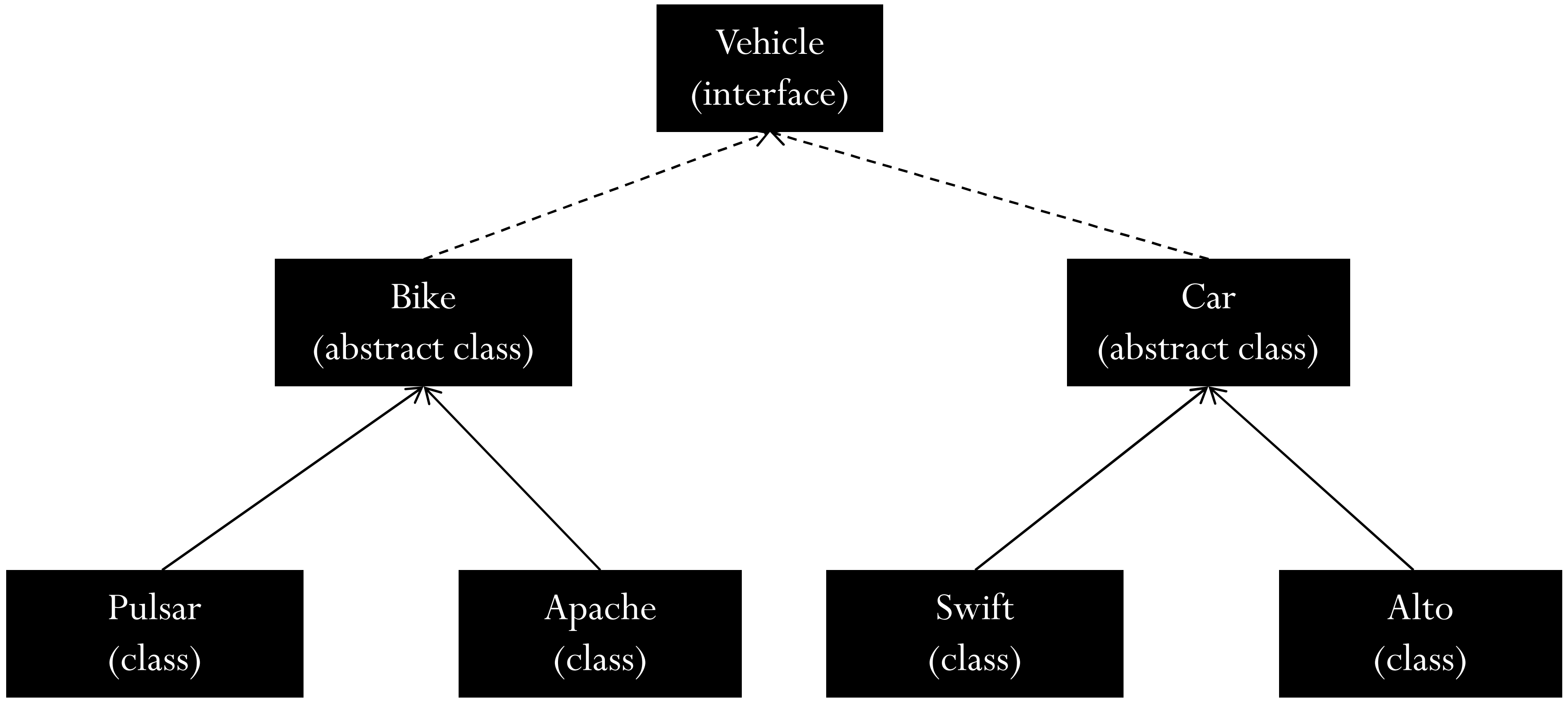
- Abstract methods must be implemented in the subclasses else they must be made abstract

Interfaces

- Interfaces are kind of abstract classes which can have
  - only abstract methods and
  - constants
- They can be implemented by classes or extended by other interfaces
- It gives you the hierarchy of classes to have different implementations

Interfaces & Abstract classes

- If a showroom wants to show vehicle information to the customer they can go with below hierarchy



Access Specifiers

- In Java we have four access specifiers where 3 have keywords and one no keyword, they are:
  - private
  - No keyword (package scope)
  - protected
  - public

Exception Handling

try, catch, finally, throw & throws

Exception Handling

- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

- There are various situations when an exception could occur:
  - Attempting to access a file that does not exist
  - Interacting with the databases

Exception Handling Keywords

Java's exception handling is managed using the following keywords: *try, catch, throw, throws and finally*.

```
try {
 // code comes here
 }
catch(TypeofException  obj) {
    //handle the exception
  }
      finally  {
           //code to be executed before the program ends
 }
```

Exception Handling Keywords (try & catch)

- Any part of the code that can generate an error should be put in the *try* block
- Any error should be handled in the *catch* block defined by the *catch* clause
- After try you can have multiple *catch* blocks

Exception Handling Keywords (throw)

- At times you may want to *throw* the exceptions explicitly which can be done using the *throw* keyword

- The general form of throw is:
  - *throw ThrowableInstance*

Exception Handling Keywords (throws)

- Sometimes methods
  - Doesn’t want to handle the exception
  - Caller knows how to handle

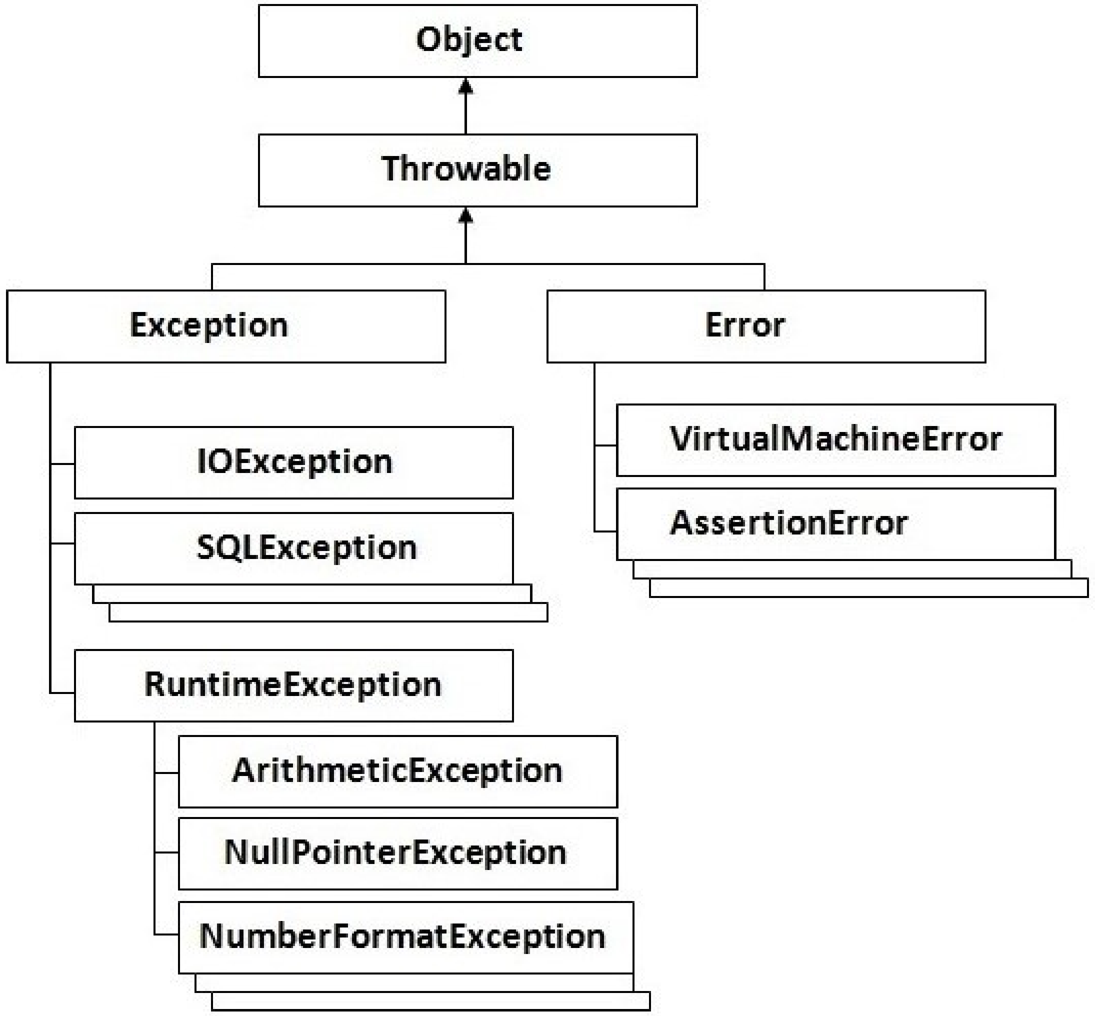
- While declaring such methods, you have to specify what type of exception it may throw by using the throws keyword

Exception Handling Keywords (finally)

- When an exception occurs, the execution of the program takes a non-linear path, and could bypass certain statements

- The *finally* block is guaranteed to execute certain statements which are mandatory

Exception Hierarchy



Checked Exceptions

- When applications are performing some operations chances of exceptions can be high
  - Database operations
  - File read/write operations

- These operations may cause exceptions for some reasons
  - Compiler always force you to handle such exceptions
  - So programs recover from the errors

Unchecked Exception

- Certain exceptions are not forced to handle
  - Those can be avoided within the program itself
  - Null check or type check

- The class RuntimeException and all its subclasses are categorized as Unchecked Exceptions

- If there is any chance of an unchecked exception occurring in the code, it is ignored during compilation

User Defined Exceptions

- Java provides extensive set of in-built exceptions

- But there may be cases where we may have to define our own exceptions which are application specific
  - UserIdNotFoundException
  - InsufficientBalanceException

- To create user defined exceptions you must subclass any of the exception classes

IO Streams

Byte & Character Streams

IO Streams

- IO Streams are the data that flows for operations like:
  - Read
  - Write

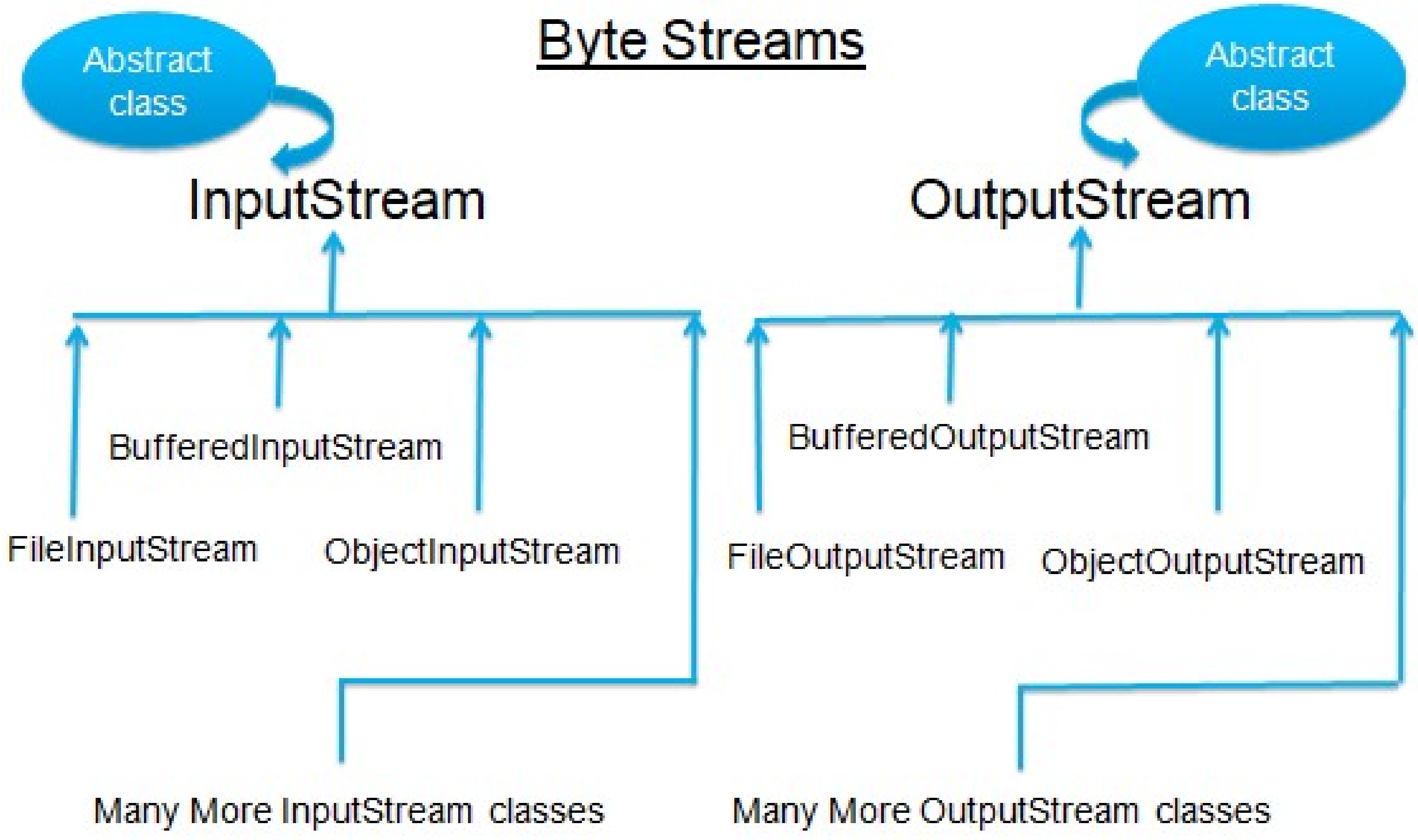
- Read/Write operations can be done on various source and destinations
  - Files
  - Buffer

IO Streams (Contd.)

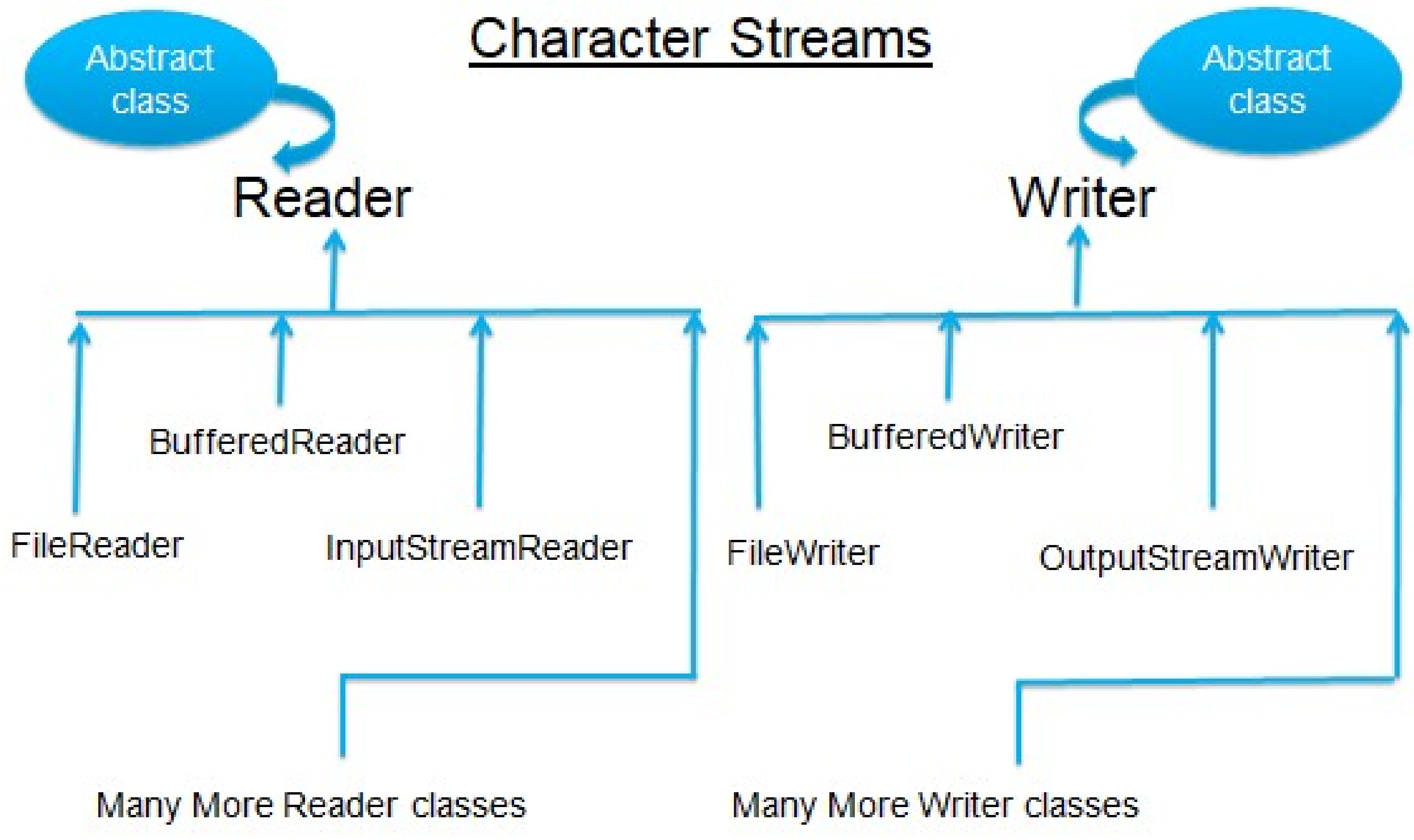
- Java’s stream classes are defined in the java.io package.

- Java defines two types of streams:
  - Byte streams
  - Character streams

IO Streams Hierarchy



IO Streams Hierarchy



Byte Stream Classes

- FileInputStream & FileOutputStream
- BufferedInputStream & BufferedOutputStream
- DataInputStream & DataOutputStream
- ObjectInputStream & ObjectOutputStream

Character Stream Classes

- FileReader & FileWriter
- BufferedReader & BufferedWriter
- InputStreamReader & ObjectStreamWriter

Multithreading

Threads, Runnable, Synchronization, Deadlocks, Fork & Join

What is Multithreading?

- In Multithreading,
  - Thread is the smallest unit of code

- A single program can perform two tasks using two threads

- Only one thread will be executing at any given point of time given a single-processor architecture

Multitasking vs Multithreading

- Multitasking:
  - Each process requires its own separate address space
  - Context switching takes more time
  - Processes are heavyweight

- Multithreading:
  - Threads are part of the same process
  - Context switching takes less time
  - Threads are lightweight

Uses of Multithreading

- A multithreaded application performs two or more activities concurrently

- It is accomplished by having each activity performed by a separate thread

- Threads are the lightest tasks within a program, and they share memory space and resources with each other

Creating Threads

- There are two ways to create threads
  - Thread class
  - Runnable interface

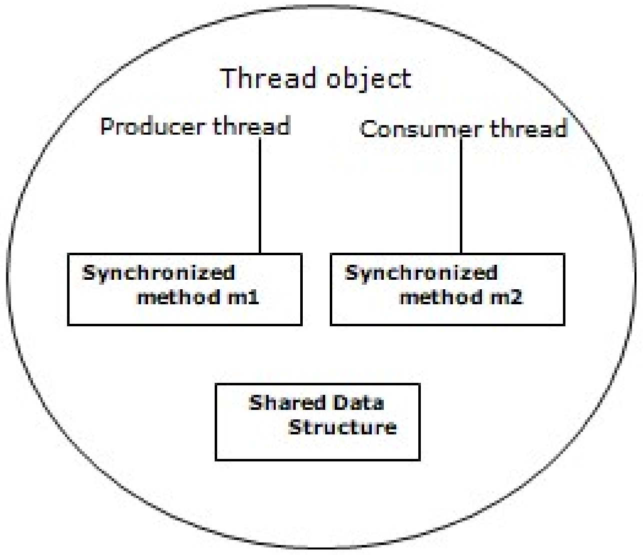
- Two important methods in multithreading
  - start
  - run

Synchronization

- It is normal for threads to be sharing objects and data
- Different threads shouldn’t try to access and change the same data at the same time
- Threads must therefore be synchronized

Synchronization (Contd.)

- This example use concurrent threads that share a common resource: a data structure.



Synchronization (Contd.)

- The current thread operating on the shared data structure, must be granted mutually exclusive access to the data

- The current thread gets an exclusive lock on the shared data structure, or a **mutex**

- A **mutex** is a concurrency control mechanism used to ensure the integrity of a shared data structure

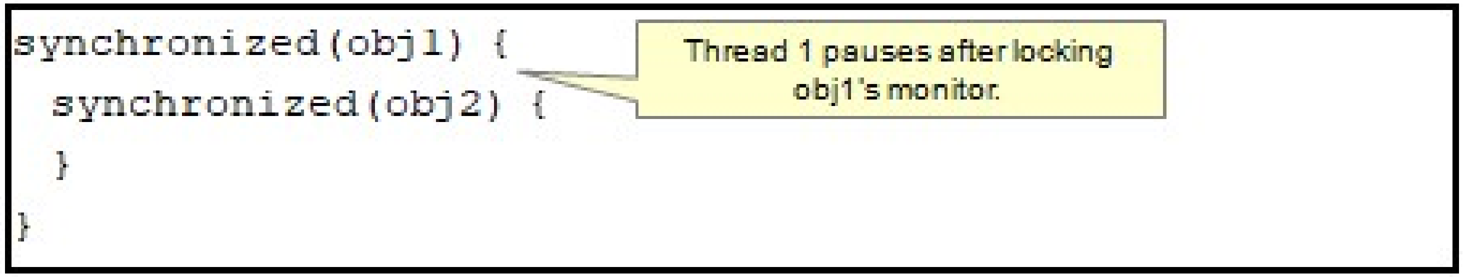
Synchronization (Contd.)

- Every object in Java has a lock, using *synchronization* enables the lock and allows only one thread to access that part of code

- Synchronization can be applied to:
  - A method
  - A block of code

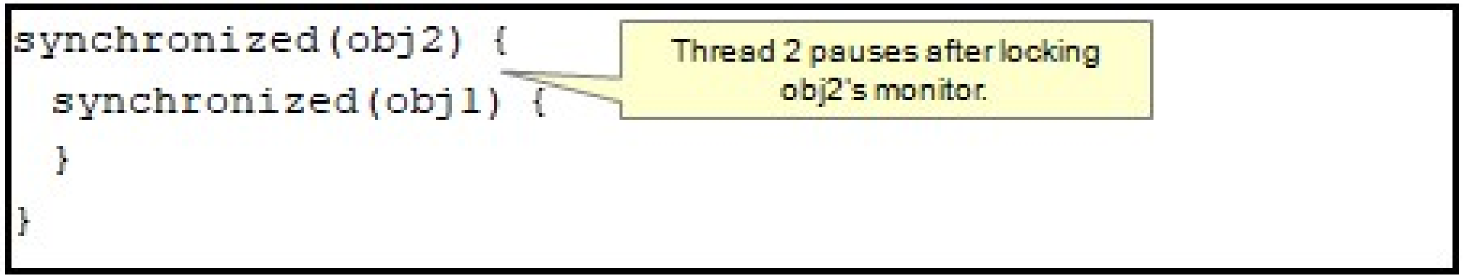
Deadlock

- Deadlock results when two or more threads are blocked forever, waiting for eachother



```
synchronized(obj1) {
  synchronized(obj2) {
  }
}
```

Thread 1 pauses after locking obj1's monitor.



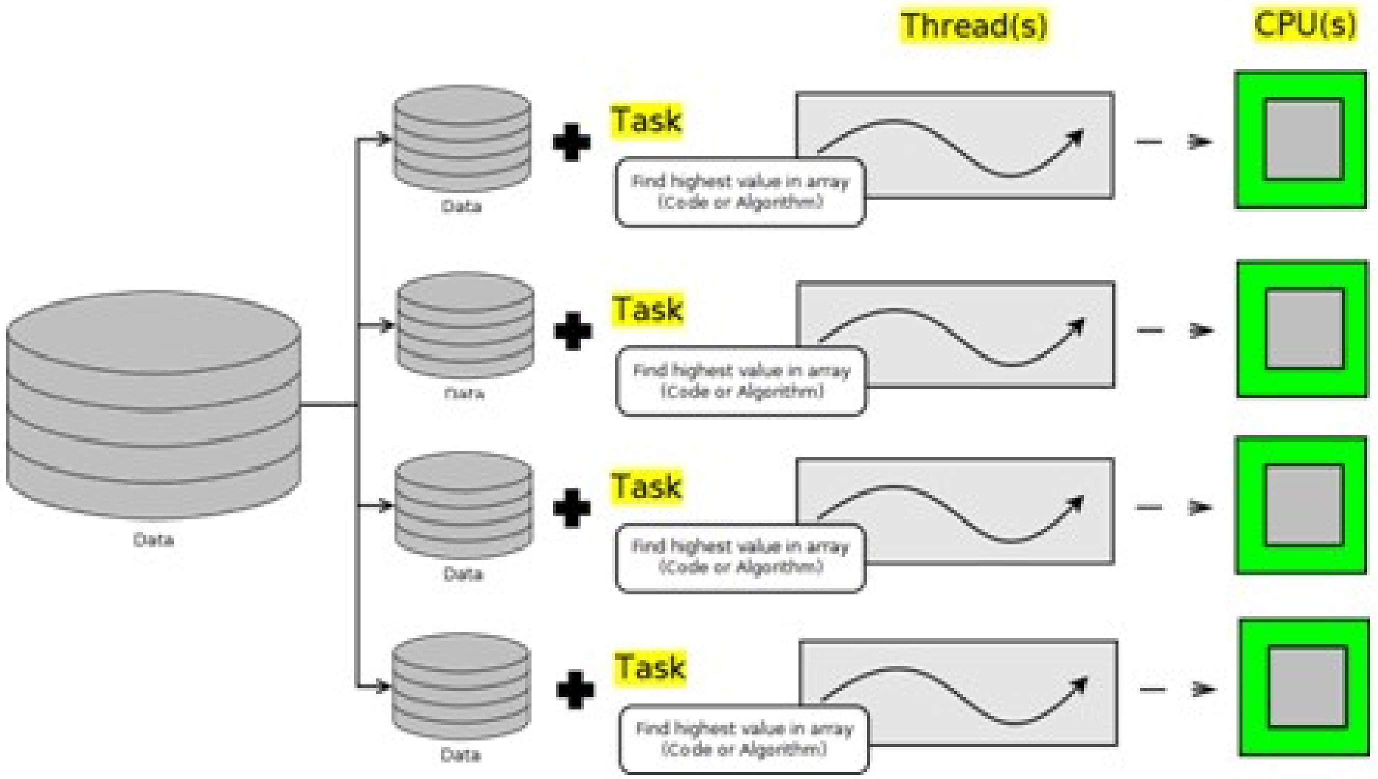
```
synchronized(obj2) {
  synchronized(obj1) {
  }
}
```

Thread 2 pauses after locking obj2's monitor.

Fork-Join

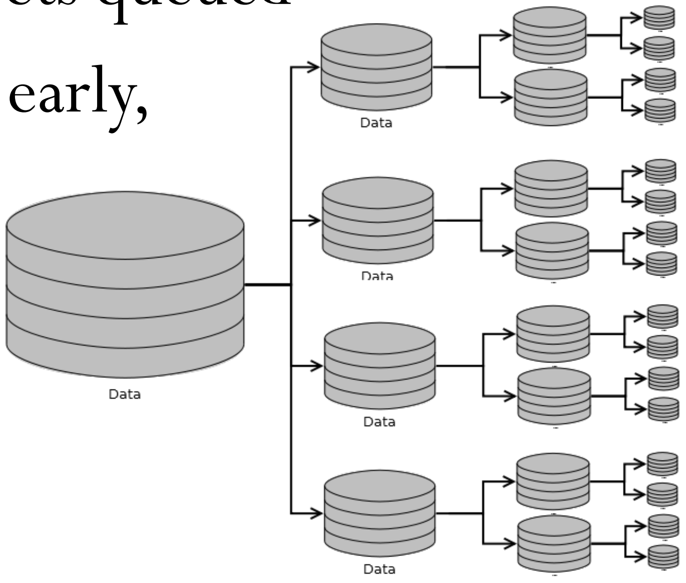
- Splitting datasets into equal sized subsets for each thread to process has a couple of problems. Ideally all CPUs should be fully utilized until the task is finished but:
  - CPUs may run at different speeds
  - Non-Java tasks require CPU time and may reduce the time available for a Java thread to spend executing on a CPU
- The data being analyzed may require varying amounts of time to process

Fork-Join (Contd.)


Thread(s)
CPU(s)
Data
Data
Task
Find highest value in array (Code or Algorithm)
Data
Task
Find highest value in array (Code or Algorithm)
Data
Task
Find highest value in array (Code or Algorithm)
Data
Task
Find highest value in array (Code or Algorithm)

Fork-Join (Contd.)

- To keep multiple threads busy:
  - Divide the data to be processed into a large number of subsets
  - Assign the data subsets to a thread's processing queue
  - Each thread will have many subsets queued
  - If a thread finishes all its subsets early, it can "steal" subsets from another thread.



Collections & Generics

List, Set, Comparable, Comparator

Collection Framework

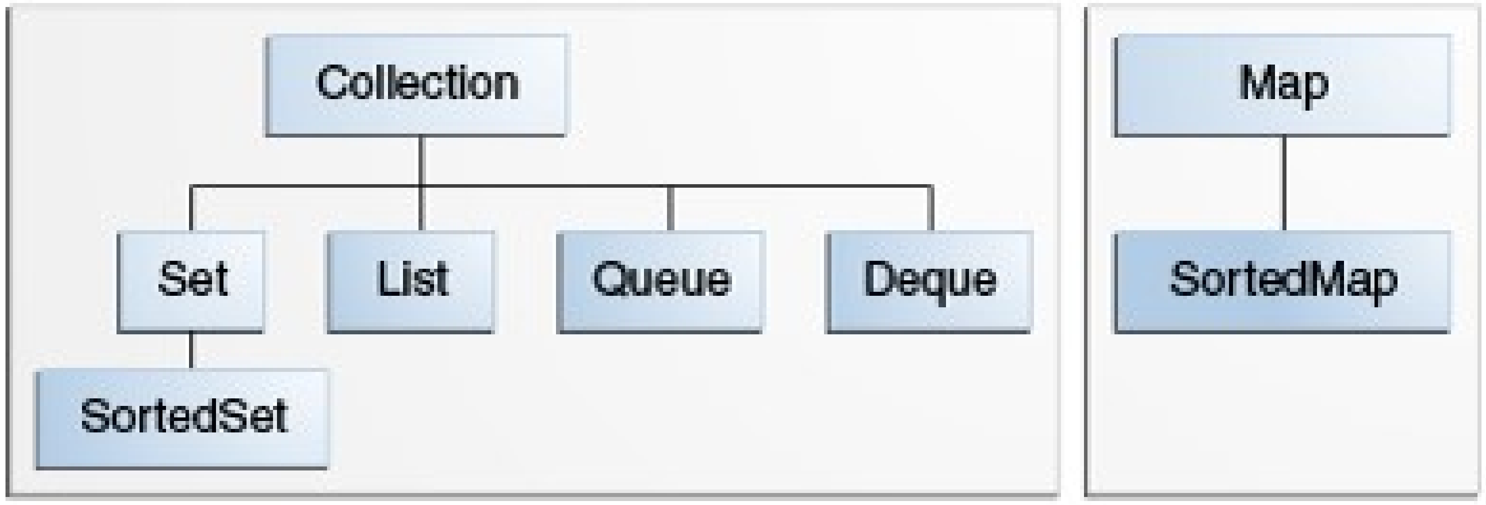
- A *collection* — sometimes called a container — is simply an object that groups multiple elements into a single unit.

- Collections are used to
  - store,
  - retrieve,
  - manipulate

What is a Collection Framework?

- A *collections framework* is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:
  - Interface
  - Implementation
  - Algorithm

Core Collection Interfaces



Collection Interface

- A Collection represents a group of objects known as its elements.

- The Collection interface contains methods that perform basic operations, such as
  - int size(),
  - boolean isEmpty(),
  - boolean add(E element),
  - boolean remove(Object element,)
  - void clear()

Set Interface

- A Set is a Collection that cannot contain duplicate elements.

- The Set interface contains *only* methods inherited from Collection and adds the restriction that duplicate elements are prohibited

- Set uses Object class methods like:
  - hashCode
  - equals

Set Implementation

- Set implementations:
  - HashSet
  - TreeSet
  - LinkedHashSet

- Object class hashCode() and equals() invoked while Set maintains the items to avoid duplicates

Generics in Collection

- Earlier Collections maintained all items as Object without generics, this let
  - Unsafe type operations
  - ClassCastExceptions

- Generics make the collection to have uniformed types in one container, so that while retrivieving you don't need to do a type check
  - *Set<T> hashSet = new HashSet<T>();*
  - *Set<T> linkedHashSet = new LinkedHashSet<T>()*

List Interface

- A List is an ordered Collection (sometimes called a *sequence*).

- A List has implementations like:
  - LinkedList
  - ArrayList

- Lists may contain duplicate elements.

List Algorithms

- Most Collections class methods are applied on List, these methods have algorithms which makes it easy to manipulate lists:
  - sort
  - shuffle
  - reverse
  - swap
  - binarySearch

Queue Interface

- A Queue is a collection for holding elements prior to processing.

- Queues typically, but not necessarily, order elements in a FIFO (first-in-first-out) manner.

- Among the exceptions are priority queues, which order elements according to their values.

Deque Interface

- It is a double ended queue

- A double-ended-queue supports
  - insertion and removal of elements at both end points
  - accessing the elements at both ends

Map Interface

- A Map is an object that maps keys to values.

- A map cannot contain duplicate keys

- The Map interface includes methods for basic operations:
  - put, get, remove, size

Map Interface methods

- Map has following methods
  - put(key, value)
  - get(key)
  - remove(key)
  - keySet()
  - entrySet()
  - values()

Map Implementation

- Map has implementations like
  - Hashtable
  - HashMap
  - TreeMap:
  - LinkedHashMap

Comparable Interface

- Comparable implementations provide
  - Natural ordering for a class
  - Has a single method compareTo which must return
    - a negative integer
    - zero
    - a positive integer

Comparator Interface

- What if you want to sort some objects in an order other than their natural ordering?
- Or what if you want to sort some objects that don't implement Comparable?
- Comparator interface consists of a single method.
  - int compare(Object o1, Object o2)

Date

- Date class is used to maintain Date, you have two Date class in Java
  - java.util
  - java.sql

- Most of the methods of java.util.Date is deprecated and you can use new Date Api provided in Java 8 i.e, LocalDate

JDBC

Java Database Connectivity

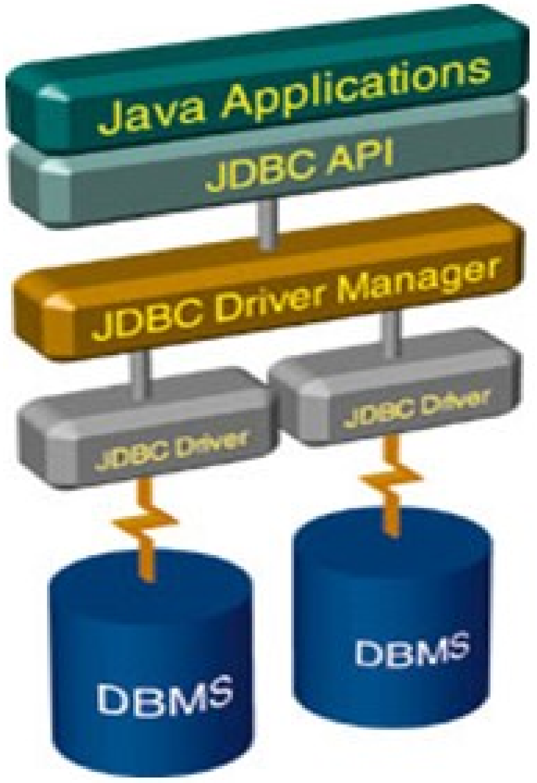
JDBC

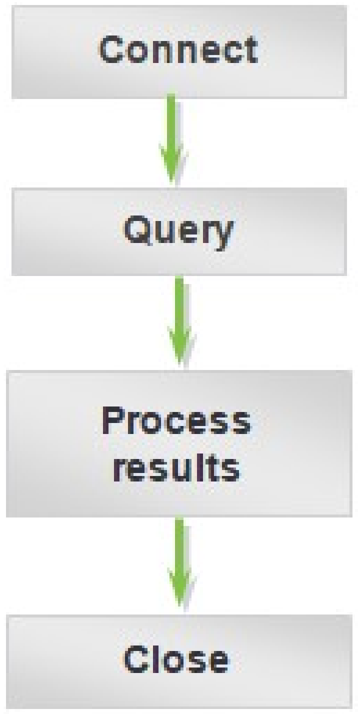
- It provides set of API's so that you can interact with any relational database

- It uses JDBC drivers to enable java programs to interact with the database

- JDBC Api's are present in java.sql.* package

JDBC Architecture





JDBC Driver

- Is a set of classes and interfaces, written according to JDBC API to communicate with a database.

- Can also provide a vendor’s extensions to the JDBC standard

Steps in JDBC

- Loading the JDBC driver
- Establishing the Connection
- Creating Statements
- Execute Queries
- Closing the resources

Prepared Statement

- Using PreparedStatement in place of Statement interface will improve the performance of a JDBC program.

- Use this object for statements you want to execute more than once

- A prepared statement can contain variables that you supply each time you execute the statement

Transactions

- With JDBC drivers:
  - New connections are in autocommit mode
  - Use conn.setAutoCommit(false) to turn autocommit off

- To control transactions when you are not in autocommit mode:
  - conn.commit()
  - conn.rollback()

ResultSetMetaData

- ResultSetMetaData is an interface which contains methods to get information about the types and properties of the columns in the ResultSet object.

- ResultSetMetaData object provides metadata, including:
  - Number of columns in the result set
  - Column type
  - Column name

RowSet

- Database Management Systems or the drivers provided by some database vendors do not support result sets that are scrollable and/or updatable.

- RowSet provides scrollability and updatability for any kind of DBMS or driver.

Types of RowSets

- RowSets are classified depending on the duration of their connection to the database
  - Connected
  - Disconnected

- After completion, it disconnects from the data source.

Types of RowSets

- CachedRowSet
- JdbcRowSet
- WebRowSet

Factory Pattern

- Factory Pattern defines an interface for creating an object, but let subclasses decide which class to instantiate.
  - Factory Method returns objects of different classes

- Factory pattern allows to
  - work on the interfaces when the developers are calling other developers code