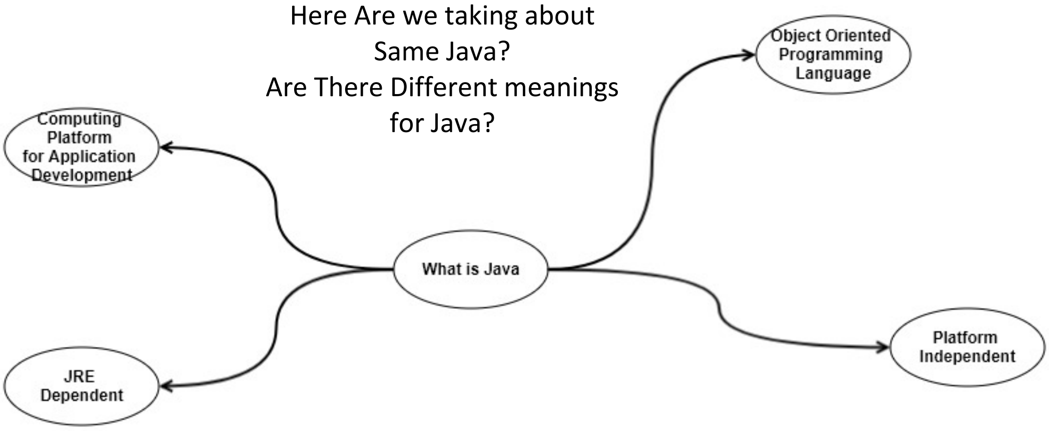
What is Java

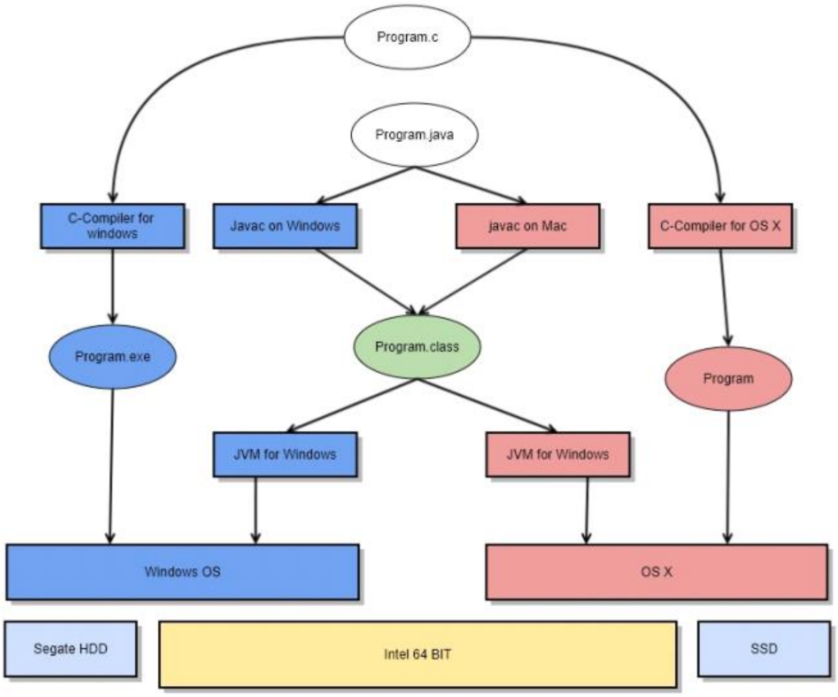
Tuesday, July 30, 2019 9:14 AM

Here Are we taking about
Same Java?
Are There Different meanings
for Java?


Computing Platform for Application Development
Object Oriented Programming Language
What is Java
JRE Dependent
Platform Independent

Java Vs C (or C++)

Tuesday, July 30, 2019 9:24 AM



Notes:

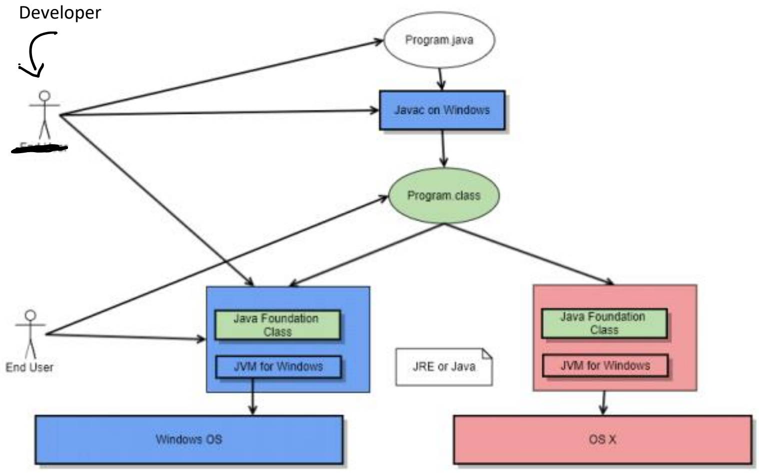
- **Legends**
  - **Blue ---> Related to windows**
  - **Pink ---> Related to Mac**
- **C/C++**
  - **Same C code can work on different platforms or virtual machines (such as windows, linux, osX**
  - **We need a compiler for particular OS. The compiler is expected to generate the code that can execute on a given OS.**
  - **Compiler for one OS wont work for other OS**
  - **Code compiled targeting one OS wont work on other OS**
- **Java**
  - **We need to compile the code once**
  - **We would use different compiler on different OS**
  - **The Different compilers are not compiling for (targeting) different OS. They simply run on different OS as an application.**
  - **The compiled code is going to be identical.**
  - **Compiler or JVM are platform dependent. Thy may application written in java platform independent**

Note:

- Even C/C++ programs are not targeting **real machine** or hardware.
- They target a particular OS.
- OS acts as a **virtual machine** for the real hardware exactly in the same way as JVM acts as virtual machine on the top of given OS

Java Distributions

Tuesday, July 30, 2019 9:38 AM



Note:

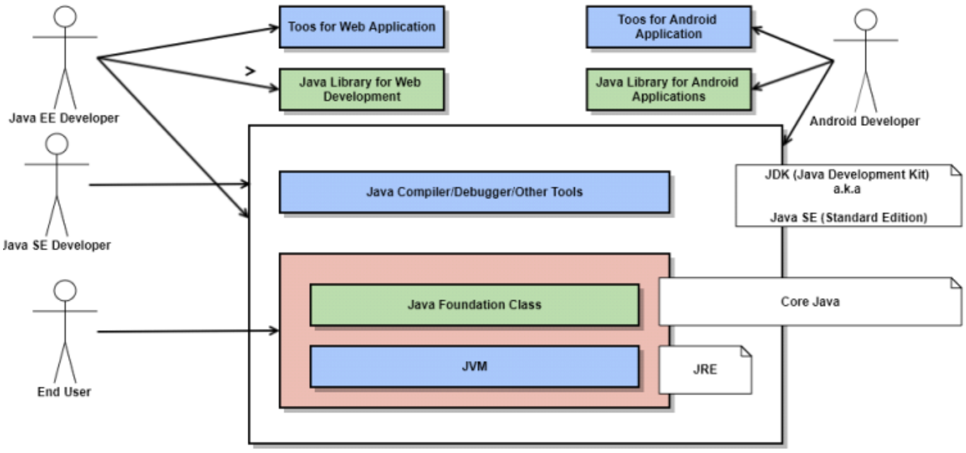
End Use Perspective

- **An End user needs only**
  - **JVM**
  - **Common Foundation Cass Library used by our application**
- **This JVM + FCL combination is referred as Java Runtime Environment (JRE) or simply Java**

Developer's Perspect

- **Developers apart from JRE also need**
  - **Compiler**
  - **Debugger**
  - **Other development related tools**
- **This bundle is referred as**
  - **JDK or Java SE**

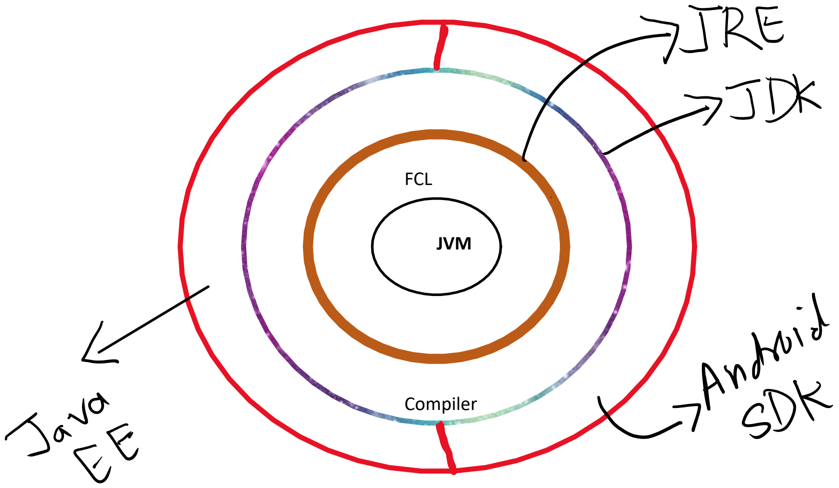
> >


Java EE Developer
Toos for Web Application
>
Java Library for Web Development
Toos for Android Application
Java Library for Android Applications
Android Developer
Java SE Developer
Java Compiler/Debugger/Other Tools
JDK (Java Development Kit) a.k.a
Java SE (Standard Edition)
Java Foundation Class
Core Java
JVM
JRE
End User

>

Specialized Variations

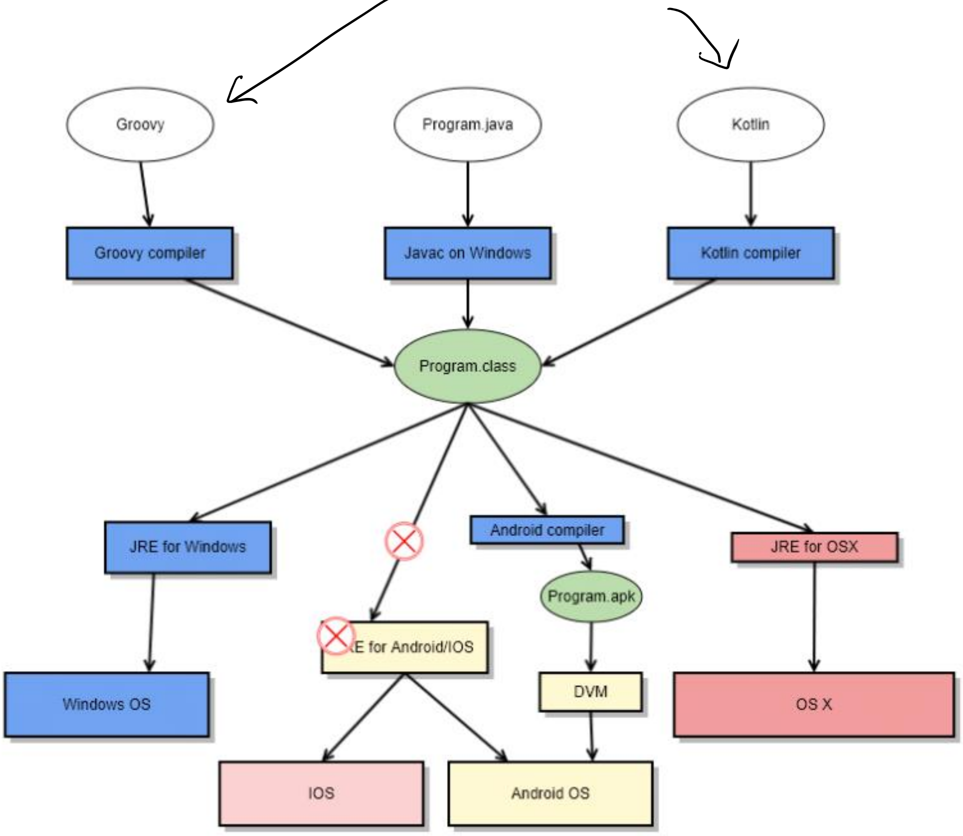
- **Apart from SE we can have specialized extensions for different application needs**
- **Enterprise Java is a set of specialized tools for creating distributed enterprise level application**
- **Android SDK is a set of tools and libraries for developing Android based Application**



There Are Two Java

Tuesday, July 30, 2019 9:48 AM

Java Language competitors



1. **Java - The Language**

- **The Object Oriented Language which is expected to run on Java Platform**
- **The language is Platform Independent**
- **Comparable to C# language of .NET platform**
- **Today This language has competitors that can be compiled to run on Java Platform**

2. **Java - the Platform**

- **This component is responsible for**
  1. **Running Java Application**
  2. **Making it platform independent**
- **Java can run on only those device that have Java Platform installed**

There is No True Java Platform (JRE) available on mobile devices like IOS or Android

Android Uses Java (language and probably its competitors) and the modifies the byte code to run on DVM rather than JVM

Java Source File

Tuesday, July 30, 2019 9:14 AM

1. A source file should have .java extension
2. A source file name is case sensitive
3. A source file can contain one or more classes

Packages

Tuesday, July 30, 2019 12:23 PM

1. Packages group related classes
2. A package is associated with a folder on the drive
   a. A folder is not a java element
   b. A package is a recognized java element
3. A class marked need to specify its package using a package statement
4. A package statement (if present) must be the first statement in a file
   a. **package furnitures;**
5. Your .java file must be present in a folder matching package name
6. You classpath should include the folder containing your .java/.class or packages
   a. Package folder shouldn't be part of classpath
7. Eleements define within a package is by default accessible only within the package
8. To access any class/method/fields from classes outside current package you must make them **public**
9. Conventionally a package name must be all lowercase.
10. When a class doesn't contain a **package** specification, it is considered to be part of **default global** package

Accessing a packaged element

1. Options 1: Use class qualified names
   - console.Input kb=new console.Input();

Non-Static or Object level members

Tuesday, July 30, 2019 4:45 PM

- Non Static Fields
  - Belongs to individuals object
  - Each object will have its own unique copy of those fields
- Non Static Methods
  - Defines Object's behavior
  - Can access both static and non-static fields and methods
  - Can be accessed only using the object reference

Static Members (a.k.a Shared Members a.k.a class level members)

Tuesday, July 30, 2019 4:41 PM

- **Static Fields**
    - **Contains a single copy of information that is shared among all the objects of a class**
    - **Every object of the class can access it and modify it as if it is its personal copy**
    - **Changes done by an object will reflect in every other object**
    - **Used common sharable information**
        - **E.g. InterestRate is likely to be same for all objects of the class**
        - **E.g. All Card in a deck will have same background.**
- **Static Methods**
    - **can be access using either**
        - **Class Reference**
        - **Object Reference**
    - **doesn't need an object reference (it can use it)**
    - **Can access only other static members (methods or fields)**
    - **Can't directly access the non static members (methods or fields)**

Static Vs NonStatic

Wednesday, July 31, 2019 9:26 AM

```
class BankAccount{

        ~~static int lastId=0;~~
        ~~static double interestRate=12;~~

        int accountNumber;
        double balance;
        String password;

        public void deposit(double amount){

        }
}

public static void main(String []args){

        BankAccount a1=new BankAccount("p@ss",1000);
        BankAccount a2=new BankAccount("w0rd",2000);

}
```

```
class Bank{

        ~~static~~ int lastId=0;
        ~~static~~ double interestRate=12;

        BankAccount [] accounts;

        int openAccount(String password, double amount){
                ...
                int accountNumber=++lastId;
                BankAccount a=new BankAccount(accountNUmber,password,amount);

                //add this account to the bank's accounts collection

                return accountNumber;
        }

        public void deposit(int accountNumber, int amount){

                BankAccount a= findAccount(accountNumber);

                a.deposit(amount);

        }


}

public static void main(String []args){

        ~~BankAccount a1=new BankAccount("p@ss",1000);~~
        ~~BankAccount a2=new BankAccount("w0rd",2000);~~


        Bank icici=new Bank();

        int a1= icici.openAccount("p@ss",1000);
        int a2=icici.openAccount("w0rd",2000);


        icici.deposit(a1, 2000);
}
```

Problem with main() tests

Thursday, August 1, 2019 11:22 AM

1. We test many functions
2. we display output on screen.
    a. It is difficult to partition which output is associated with which test
    b. It is difficult to understand if the output is what we expected
3. Execution of one test may influence the outcome of other test
    a. Tests are not isolated
4. There is no clear definition if a given test is giving expected result
5. There is no way of ensuring that we have tested all possible outcome of a function

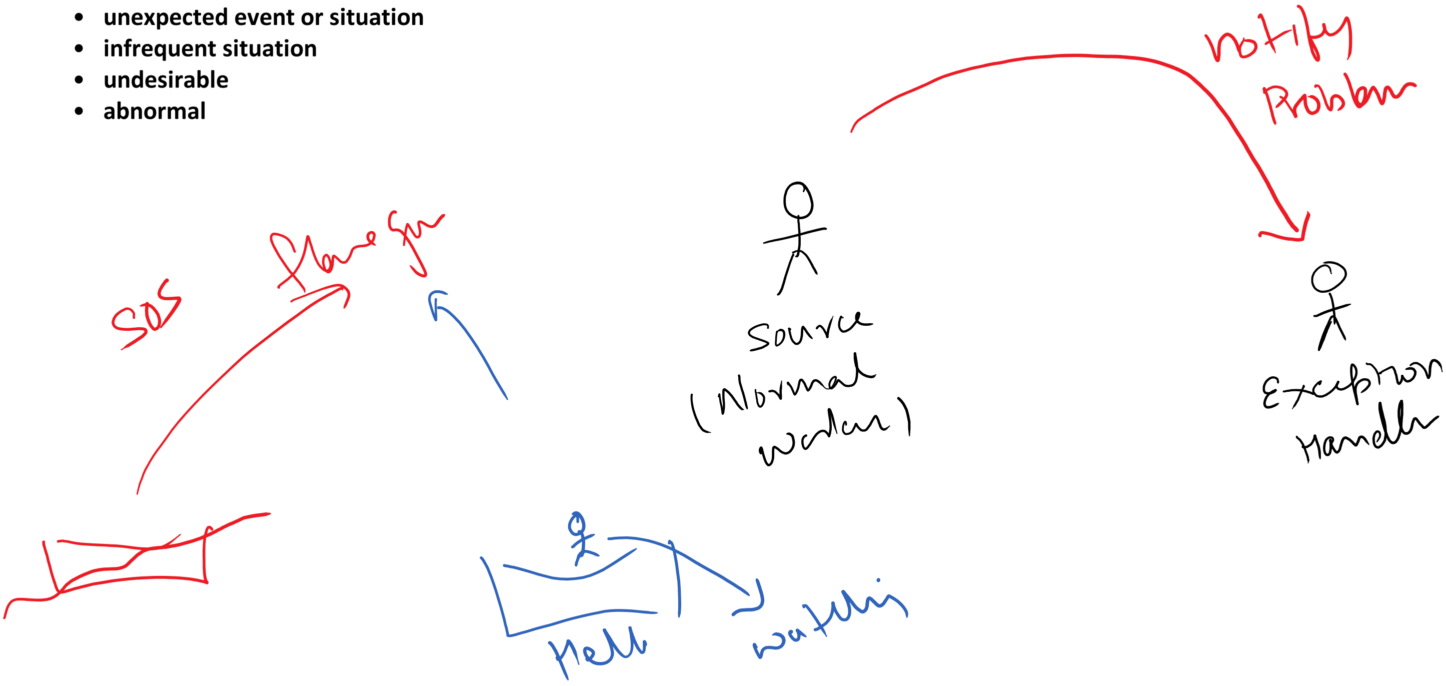
Exception Handling

Saturday, August 3, 2019 1:49 PM

1. What is Exception?
   - Runtime Error
   - Event that stops the current execution
   - Interrupts the execution of the program
   - Abnormal termination of the program

Not the definition of exception. It's the consequence of not handling it.

- **unexpected event or situation**
- **infrequent situation**
- **undesirable**
- **abnormal**



Exception Handling Elements

Saturday, August 3, 2019 2:24 PM

Exception

- undesirable situation
- has potential to break your application's normal life cycle

Exception Handling

- Attempt to solve the problem
    - and put program back to track
    - make a graceful exit if there is no solution

Steps of Exception Management

1. Identifying the situation
    a. You must know there is a problem
2. Sending a signal to the handler
3. Handler recognizing the signal
4. Handler taking corrective measure

Exception Handling Traditional Approach

Saturday, August 3, 2019 2:24 PM

Steps of Exception Management

1. Identifying the situation
    - ○ you check for conditions in **if statement**
2. Sending a signal to the handler
    - **return** a value indicating the problem
        - Popular values include
            - □ false
            - □ 0
            - □ null
            - □ -1

3. Handler recognizing the signal
    - use **if** statement to check if error has occurred

4. Handler taking corrective measure
    - write code in if block or else block

Problems with Traditional approach

1. no standard value to indicate error
    a. you may use boolean, null, -1, 0
2. often meaning or details of exception is lost
    a. transfer() returns false. whats the reason for failed transfer
        i. invalid credential
        ii. insufficient balance
        iii. invalid account info
3. exception should be passed from the point of origin to point of handling
    a. In each step the intermeditary function must check the error and pass to the next level

Exception Handling OO Approach

Saturday, August 3, 2019 2:24 PM

Steps of Exception Management

- Exception is **a special object**
  - This object is different from information such as false, null or 0
- The object may act as a signal
- The object may contain details for the problem
  - and possible resolution

Steps of Exception Management

1. Identifying the situation
   - you check for conditions in **if statement**
2. Sending a signal to the handler
   - **we throw rather than ~~return~~** a value indicating the problem
   - throw is different from return.
     - this makes code clear, if we are handling business condition or error condition
   - **throw is not hand me down**
   - throw **automatically propagates** from point of origin to point of handling
   - throw automatically by passes all calls till it gets a **matching catch**

3. Handler **catch** the **exception**
   - Handles the exception

4. **Try**
   - is preparing for the exception
   - you need to be ready **(try)** to catch
   - a **catch** must be associated with a try
   - **A try** should be followed by one or more catch
   - **Remember: try is not associated with throw. Its associated with catch**

Exception Handling OO Approach

Saturday, August 3, 2019 2:24 PM

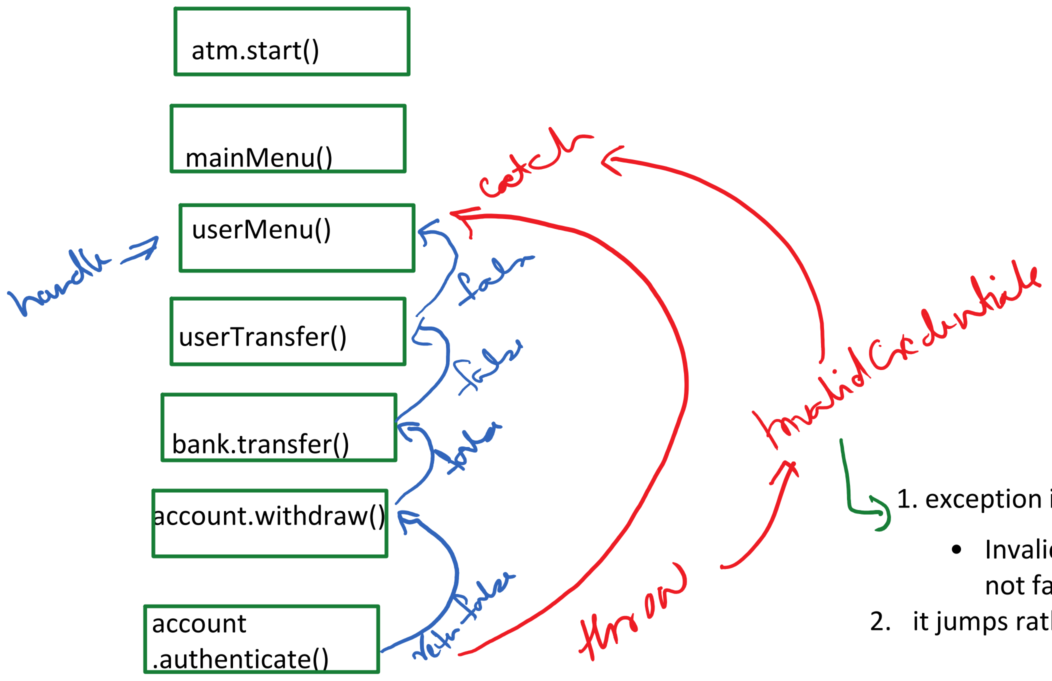
Steps of Exception Management

- Exception is **a special object**
  - This object is different from information such as false, null or 0
- The object may act as a signal
- The object may contain details for the problem
  - and possible resolution

Steps of Exception Management

1. Identifying the situation
   - you check for conditions in **if statement**
2. Sending a signal to the handler
   - **we throw rather than ~~return~~** a value indicating the problem
   - throw is different from return.
     - this makes code clear, if we are handling business condition or error condition
   - **throw is not hand me down**
   - throw **automatically propagates** from point of origin to point of handling
   - throw automatically by passes all calls till it gets a **matching catch**

3. Handler **catch** the **exception**
   - Handles the exception



1. exception is meaningful
   - InvalidCredential and not false
2. it jumps rather than return

Exception Related Elements

Saturday, August 3, 2019 1:54 PM

try

- is a block of code that calls functions that may throw exception
- generally we don’t throw exception directly in the try block
- try is associated with one or more catch block (and optionally a final block)

catch

- handles an exception
- A catch is always associated with a try
- we may have multiple catch block following a try
- Once an exception reaches try, it is assumed to be resolved. It doesn't propagate any further
- We may rethrow (Same or different exception) from catch block
- A catch block may catch a range of exception hierarchy
    - A catch block catching super class exception also catches sub class exception
    - catch(Runnable r) catches every single exception
    - If you need to catch both sub class (specific) and super class (generic) exception you must catch subclass exception (specialized) before catching a generalized super class exception.
- A catch block may also catch multiple unrelated exception. You may pass name of each exception class separated with a pipe character

```
try{
        …
}catch( IOException | ClassNotFoundException ex){ //can catch multiple exception

}
catch(IneterruptedException ex){  //can have multiple catch blocks

}
catch( RuntimeException ex){ //must be caught before super class Exception

}
catch(Exception ex){ //must be caught after all exception and before Throwable

}
```

throw

- throw throws the exception object
- statement following throw is ignored
- throw propagates exception by passing regular function call till it reaches matching catch block (or a matching super catch block)
- If there is no catch defined for this exception, it reaches JVM and JVM stops execution of current process and logs the exception details

finally

- In try-catch there is no guarantee that all parts of function will execute
    - If No exception occurs --> catch block is skipped
    - If An exception occurs

        - and catch block is defined
            - □ rest of try is skipped
            - □ other catch blocks (if any) is skipped
        - and catch block is not defined
            - □ rest of try is skipped
            - □ other catch blocks (if any) is skipped
            - □ code following try-catch is skipped

- What if we need to ensure one piece of code that will always excute?
    - Such codes are generally cleanup code
- we can write such codes in finally block
- A finally block is associated with a try block
- there can be only one finally following try
    - try-finally
    - try-catch-catch-catch-finally
- A finally is called if
    1. No exception occurs
        - try --> finally --> rest of the code -->normal return
    2. If exception occurs and caught
        - try (rest part skipped) -->catch --> finally --> rest of the code -->normal return
    3. If exception occurs and not caught
        - try(rest part skipped) --> finally -->(rest of code skipped) --> throw to caller function

Checked and Unchecked exception

- In java there are two types of exceptions --- Checked and Unchecked
- Both exceptions follow the above rules of throw-try-catch-finally

Checked Exceptions AND THROWS

- compiler checks if an Exception is thrown it is caught or not?
- A checked exception must either
    1. be handled using a **catch**
    2. or declared and *not handled using* ***throws declaration***
- ***throws*** *is a part of function signature*
- *It indicates that the current function may throw an exception which is not handled*
- *Any function invoking current function must either*
    - *handle the exception as* ***catch***
    - *or inform it caller about unhandled exception using* ***throws***

```
public void authenticate( String password) throws InvalidCredentialException{
        if(!password.equals(this.password))
                throw new InvalidCredentialException(); //not caught here
}

public void withdraw(int amount , String password) throws InvalidCredentialException,
InsufficientBalanceException{
        authenticate(password);
        if(amount>balance)
                throw InsufficientBalanceException();
}

//the handler
```

```
void atmMenu(){
        try{
                withdraw(100,"pass");
                dispenseCash(100);
        }catch(BankingException ex){
                //handles both the above exceptions
                //no need to throws
        }
}
```

Unchecked Exception

- doesn't make catch or throws mandatory
- Developers have a choice to catch exception or completely ignore it

Recommendation

- avoid checked exceptions
- Modern application frameworks prefer uncheck exception over checked exception.

Generic Collection

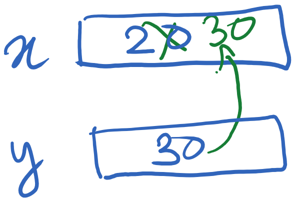
Saturday, August 3, 2019 3:01 PM

Value Type Vs Reference types

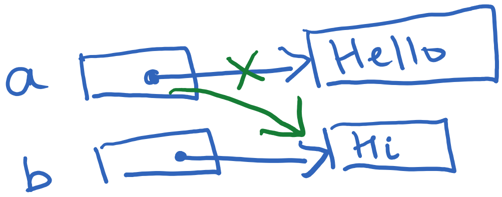
Monday, August 5, 2019 3:25 PM

- Java defines primitive data types (int, float, boolean) as value type
- Classes / Objects are reference type
- There memory allocation is different

```
int x=20;

int y=30;

x= y;
```



```
String a="Hello";

string b="Hi"

a=b;
```



Primitive Wrapper Classes

Monday, August 5, 2019 3:30 PM

- primitive types are not subclass of Object
  - int intanceof Object <--- wrong
- To make int work as Integer we have wrapper classes

int	java.lang.Integer
float	java.lang.Float
double	java.lang.Double
boolean	java.lang.Boolean

How to Use

```
//explicit approach

Integer i=new Integer(7);

int j= i.intValue();


//implicit autobox-unbox approach

Integer i= 7;  //autobox to new Integer(7)

int j= I ; //auto unbox to i.intValue()
```