Day 13

Sole constructor

- A constructor of super class, which is designed to call from constructor of sub class is called sole constructor.

```
abstract class A{
    private int num1;
    private int num2;
    public A(int num1, int num2) {  //Sole Constructor
        this.num1 = num1;
        this.num2 = num2;
    }
}
class B extends A{
    private int num3;
    public B(int num1, int num2, int num3 ) {
        super( num1, num2 );    //Call to suprt class constructor
        this.num3 = num3;
    }
}
public class Program {
    public static void main(String[] args) {
        B b = new B( 10,20,30);
        b.printRecord();

    }
}
```

toString

- toString is non final / non native method of java.lang.Object class
- Syntax: "public String toString( )"
- Use

```
Employee emp = new Employee( ... );
String str = emp.toString( );
```

- If we want to return state of current instance in String format then we should use toString method.
- If we do not define toString method inside class then its super class toString method gets called.
- Consider source code of toString method from Object class:

```
public String toString() {
    return this.getClass().getName() + "@" +
```

```
Integer.toHexString(this.hashCode());
}
```

- The toString method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object.
- In short, toString method of Object class returns String in following format

```
F.Q.ClassName@Hashcode
```

- According to clients requirement, if implementation of super class method is partially complete then we should override method in sub class.

```
@Override
public String toString() {
    return this.name+" "+this.empid+" "+this.department+"
"+this.designation+" "+this.salary;
}
```

- The result in toString method should be a concise but informative that is easy for a person to read.

```
@Override
    public String toString() {
        return String.format("%-20s%-4d%-10.2f", this.name, this.empid,
this.salary);
    }
```

- It is recommended that all subclasses override toString() method.

equals

- If we want to compare value/state of variable of value type then we should use operator ==.

```
public static void main1(String[] args) {
    int num1 = 10;
    int num2 = 10;
    if( num1 == num2 )
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output : Equal
}
```

- In Java, primitive types are not classes. Hence we can not use equals method to compare state of variable of value type.

```
public static void main(String[] args) {
    int num1 = 10;
    int num2 = 10;
    if( num1.equals( num2 ) )   //Not OK
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output : Equal
}
```

- We can use operator == with varibale of reference type.
- If we use operator == with variable of reference type then it doesn't compare state of instances rather it compares state of reference variable.

```
class Employee{
    private String name;
    private int empid;
    private float salary;
    public Employee(String name, int empid, float salary) {
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }
}
public class Program {
    public static void main(String[] args) {
        Employee emp1 = new Employee("Sandeep",33, 45000);
        Employee emp2 = new Employee("Sandeep",33, 45000);
        if( emp1 == emp2 )//Comparing state of references
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Not Equal
    }
}
```

- If we want to compare state of instanes then we should use equals method.
- equals is non final and non native method of java.lang.Object class.
- Syntax: "public boolean equals(Object obj)"
- if we do not define equals method inside class then its super class's equals method gets called.
- Consider code of equals method of object class:

```
public boolean equals(Object obj) {
    return (this == obj);
```

```
}
```

- equals method of Object class do not compare state of instances. It compares state of references.

```
public class Program {
    public static void main(String[] args) {
        Employee emp1 = new Employee("Sandeep",33, 45000);
        Employee emp2 = new Employee("Sandeep",33, 45000);
        if( emp1.equals(emp2) ) //Calling Object class's equals method
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Not Equal
    }
}
```

- Inside equals method, if we want to compare state of instances then we should override equals method inside sub class.

```
class Employee {
    private String name;
    private int empid;
    private float salary;
    public Employee(String name, int empid, float salary) {
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }
    //Employee this = emp1;
    //Object obj = emp2;    //Upcasting
    @Override
    public boolean equals(Object obj) {
        if( obj != null ) {
            Employee other = (Employee) obj;    //Downcasting
            if( this.empid == other.empid )
                return true;
        }
        return false;
    }

}
public class Program {
    public static void main(String[] args) {
        Employee emp1 = new Employee("Sandeep",33, 45000);
        Employee emp2 = new Employee("Sandeep",33, 45000);
        if( emp1.equals(emp2) ) //Calling Employee class's equals method
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
```

```
        //Output : Not Equal
    }
}
```

- In Case of collection framework, to compare state of instances, we should use equals method.

Exception Handling

- Consider code in C/C++

1. Compiler error

```
int main( void )
{
    return 0    //Compiler error – ; missing
}
```

- If we make syntactical mistake in a program then compiler generates error.

2. Linker Error

```
int main( void )
{
    void print( );  //Local Function Declaration;
    print( );   //Function Call     =>  Linker error
    return 0;
}
```

- Without definition if we try to access any member of the class then linker generates error.

3. Bug

```
int main( void )
{
    int number = 10;
    if( number % 2 == 0 )
        printf("Even\n");
    else;   //bug
        printf("Odd\n");
    return 0;
}
```

- Logical error / syntacticaly valid but logical invalid statement is called bug.

4. Exception

- Runtime error is called exception

- It gets generated due to wrong input.

Resource type and resource

- AutoCloseable is interface declared in java.lang package.
- "void close()throws Exception" is a method of AutoCloseable interface(I/F).
- Closeable is interface declared in java.io package.
- "void close()throws IOException" is a method of Closeable interface.
- Consider following code

```
//class Test is Resource Type
class Test implements AutoCloseable{
    public Test() {
    }
    @Override
    public void close() throws Exception {
    }
}
public class Program {
    public static void main(String[] args) {
        Test t = new Test( );
        //Test t;   => reference
        //new Test( );  => Instance : resource
    }
}
```

- In context of exception handling, a class which implements AutoCloseable or Closeable interface is called resource type and its instance is called resource.

Operating system resources for java application

```
1. Thread
2. File
3. Socket
4. Connection
5. IO devices
```

What is exception

- Runtime error is called exception.
- Exception is an instance that is used to send notification to the end user of the system if any exceptional situation/condition occurs in the program.
- Operating system resources are limited hence we should use it carefully. If we want to handle it carefully then we should use exception handling mechanism.

How to handle exception

- Using following keywords, we can handle exception.
    1. try
    2. catch
    3. throw
    4. throws
    5. finally
- Throwable is a class declared in java.lang package.
- The Throwable class is the superclass of all errors and exceptions in the Java language.
- Only objects that are instances of Throwable class (or one of its subclasses) are thrown by the JVM or can be thrown by the Java throw statement.
- Similarly, only this class or one of its subclasses can be the argument type in a catch clause.

Constructors of Throwable class

1. public Throwable()

```
Throwable t1 = new Throwable( );
```

2. public Throwable(String message)

```
Throwable t1 = new Throwable("Message");
```

3. public Throwable(Throwable cause)

```
Throwable cause = new Throwable("Message");
Throwable t1 = new Throwable( cause );
```

4. public Throwable(String message, Throwable cause)

```
Throwable cause = new Throwable();
Throwable t1 = new Throwable( "Message", cause );
```

Methods of Throwable class

1. public String getMessage()
2. public Throwable getCause()
3. public void printStackTrace()

Error

- It is a sub class of java.lang.Throwable class which is declared in java.lang package.
- if runtime error gets generated due to environmental condition then it is called error

- Example:
    - InternalError
    - VirtualMachineError
    - Class OutOfMemoryError
    - StackOverflowError
- We can write catch block to handle error. We can not recover from error hence it is not recommended to write try catch block to handle error.

Exception

- It is a sub class of java.lang.Throwable class which is declared in java.lang package.
- if runtime error gets generated due to application then it is called exception
- Exception
    - NumberFormatException
    - ClassCastException
    - NullPointerException
    - ArrayIndexOutOfBoundsException
- We can write catch block to handle error. We can recover from exception hence it is recommended to write try catch block to handle exception.

Types of exception

```
1. Checked Exception
2. Unchecked Exception
```

- These types are designed for java compiler.

Unchecked Exception

- java.lang.RuntimeException and all its sub classes are considered as unchecked exception classes.
- It is not mandatory to handle unchecked exception.
- Example:
    1. NumberFormatException
    2. ClassCastException
    3. NullPointerException
    4. ArrayIndexOutOfBoundsException

Checked Exception

- java.lang.Exception and all its sub classes except java.lang.RuntimeException(and its sub classes) are considered as Checked exception.
- It is mandatory to handle checked exception.
- Example:
    1. CloneNotSupportedException
    2. ClassNotFoundException
    3. InterruptedException

4. IOException

try

- It is a keyword in Java.
- If we want to inspect group statements for excetion then we should use try block/handler.
- Try block must have at least once catch block/finally block/reource.

catch

- It is a keyword in Java.
- If we want to handle exception them we should use catch block/catch handler.
- try block may have multiple catch block.
- Only Throwable class or one of its subclasses can be the argument type in a catch clause.
- Multi catch block can handle multiple specific exception inside single catch block
- Consider Following Code

```
NullPointerException ex1 = new NullPointerException( ); //OK
RuntimeException ex2 = new NullPointerException( ); //OK => Upcasting
Exception ex2 = new NullPointerException( ); //OK => Upcasting
```

- Consider Following Code

```
InterruptedException ex1 = new InterruptedException( ); //OK
Exception ex2 = new InterruptedException( ); //OK
```

- Exception class reference variable can contain reference of checked as well as unchecked exception. Hence to write generic catch block we should use java.lang.Exception.

```
try{
    //TODO
}catch( Exception ex ){ //generic catch blocks
    ex.printStackTrace( );
}
```

throw

- It is a keyword in Java.
- In Java application, JVM/programmer can generate exception. To generate exception, we should use throw keyword.
- Only objects that are instances of Throwable class (or one of its subclasses) are thrown by the JVM or can be thrown by the Java throw statement.
- throw statement is a jump statement.

finally

- It is a keyword in Java.
- If we want to release local resources then we should use finally block.
- JVM always execute finally block.
- try block may have multiple catch block but it can contain only one finally block.

try with resource

```
try( Scanner sc = new Scanner(System.in); ) {   //try with resource
    System.out.print("Num1  :   ");
    int num1 = sc.nextInt();
    System.out.print("Num2  :   ");
    int num2 = sc.nextInt();
    if( num2 == 0 )
        throw new ArithmeticException("Divide by zero exception");
    int result = num1 / num2;
    System.out.println("Result  :   "+result);
}catch( Exception ex ) {
    ex.printStackTrace();
}
```

throws

- It is a keyword in Java.
- If we want to forward/delegate exception from one method to another method then we should use throws keyword.
    - public static int parseInt(String s) throws NumberFormatException
- If method throws unchecked exception then handling it is optional.
    - public static void sleep(long millis)throws InterruptedException
- If method throws checked exception then handling it is mandatory.

```
class A extends Exception{  }
class B extends Exception{  }
class C extends Exception{  }
class Program{
    //public static void print( )throws A, B, C{    //or
        public static void print( )throws Exception{
        if( expr1 )
            throw new A( );
        else if( expr2 )
            throw new B( );
        else if( expr3 )
            throw new C( );
        else
            //TODO
    }
}
```

Topics For tomorrow

- Custom Exception
- Exception Chaining
- Exception Propagation
- Generics
- Interface