- Interface

1. Set of protocols/rules/specifications that a class needs to follow

- Marker Interface

Empty Interface is called as Marker Interface

1. Cloneable
2. Seralizable

```
interface Batter{
int getRuns();
}

interface Bowler{
int getWickets();
}
```

```
abstract class Player{
id, name,age,
matchesPlayed;

accept(){

}
toString(){
}
}
```

```
Player [] arr = new Player[11];
arr[0] = new Cricketer();
arr[0].accept();

totalMatachespalyed = 0;
total runs = 0;
total wickets = 0;
for(Player p:arr){
totalMatachespalyed += p.getmatchesPlayed();
Cricketer c = (Cricketer) p;
totalruns += c.getRuns();
totalwickets += c.getWickets();
}

case3
or(Player p:arr){
sysout(p)
}
```

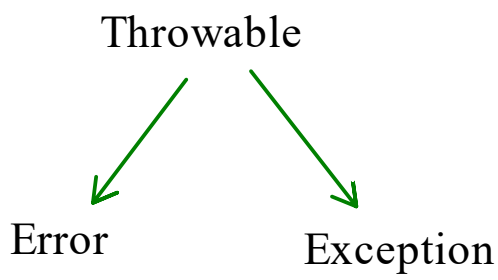
```
class Cricketer extends Player implements Batter,Bowler{
runs, wickets;

int getRuns(){
}

int getWickets(){
}

accept(){
      super.accept();
runs & wickets;
}
}
```

Exception Handling

- It is a mechanism used to handle runtime time problems

Runtime Problems

1. Problems in the code itself
2. Wrong inputs from user
3. Problems in runtime environment



1. Error

- It is recommended not to handle errors

OutOfMemoryError

StackOverFlowError

2. Exception

- It is recommended to handle exceptions
- to handle the exceptions or to perform exception handling java have provided few keywords.
- the keywords are

1. try
2. catch
3. throw
4. throws
5. finally

```
void division(int n, int d){
      cout<<"Division - "<<n/d;
}

main(){
try{
      division(10,0);
}
catch(ArithmeticException e){

}
cout<<"successfully executed"<<endl;
}
```

try
catch
throw

```
  class Exception{
  string message;

  virtual void displayStackTrace(){

  }
  }
```

```
class ArithmeticException :public Exception{


ArithmeticException(){
message = "/ by 0";
}

void displayStackTrace(){
cout<<"Arithmetic Exception"<<endl;
cout<<"Message -"<<message<<endl;
}

}
```

Exception Handling
- In java to handle the exception we can use below keywords
1. try
2. catch
- try is used to check for if their are any exceptions generated by the statements inside them.
- if any statement from the try block throw an exception then the try will look for matching catch block to handle the exception
- every try block that we provide should have atleast 1 catch block.
- a single try block can have multiple catch block.

- we can also provide a try block with the finally block.
- here the exceptions wont be handled , however tthe resources that are utilized can be closed inside this block.
- finally block executes ever time even their are exception or no any exception.

- If the resources have implemented Autocloseable interface then, we can use try-with-resource block to close such resources.

- In short we can provide a try block with
1. a single catch block
2. finally block
3. try with resource

Exceptions are of 2 types
1. Checked Exception
    - Exception class and all its sub classes except RuntimeException class are all considered as Checked Exceptior
    - These exceptions are mandatory to handle
2. UnChecked Exception
    - Runtime Exception class and all its sub classes are considered as UnChecked Exception
    - These exceptions are optional to handle.

```
class Test{

void dowork()throws IOException {
sysout("Do create the objects of date and time");
      // generate exception
      // Date
      // Time
}
}

class SubTest extends Test{

@override
void dowork()throws IOException {
      Time t1 = new Time()';
try{
      t1.sethrs(30);
}catch(InvalidTimeException e){
      throw new IOException(e);
}
}
```

Shallow Copy
Deep Copy