Object orientated programming

Encapsulation, object, cloning, exception handling

Encapsulation

- **Encapsulation** is a *process of wrapping code and data together into a single unit*, for example, like a capsule which is mixed of several medicines.
- We can create a fully encapsulated class by making all the data members of the class private.
- Setter and getter methods are needed to set and get the data in it.
- Why do we want to use it?

Advantage of encapsulation

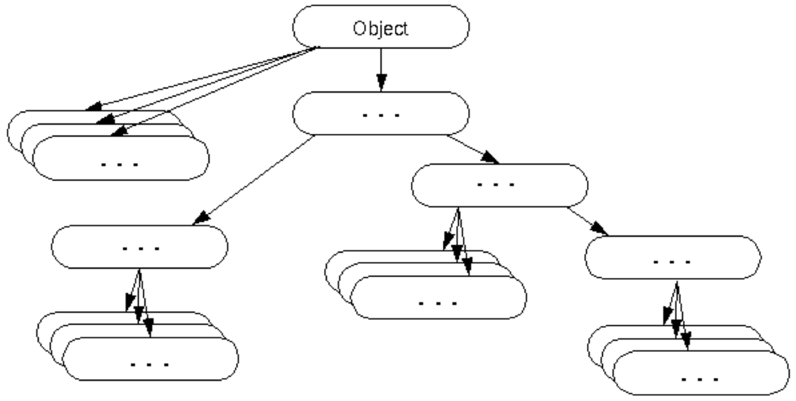
- By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods to protect different part if needed.
- It provides **control over data**.
  - Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
- It is a way to achieve **data hiding.**
  - Other classes will not be able to access the data through the private data members.
- Standard IDE's are providing the facility to generate the getters and setters. Therefore, it is **easy and fast to create an encapsulated class**.

Simple example of encapsulation

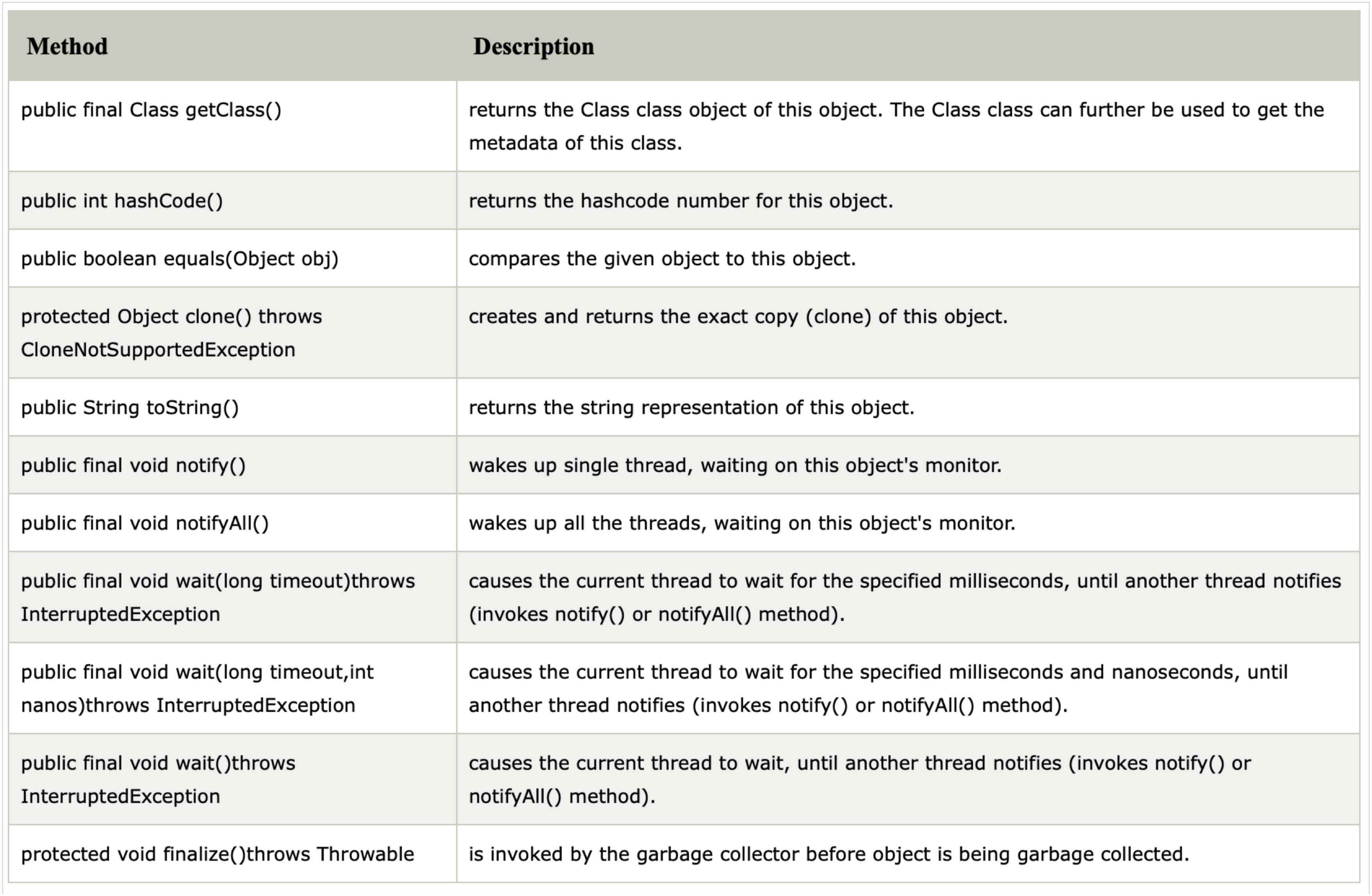
- Student example with encapsulation (com.pgr103.encapsulation)
- Read only class has only getter methods
- Write only class has only setter methods
- Another example: Account (account and testaccount)

Object class

- **The Object class** is the parent class of all classes by default.
    - In other words, it is the topmost class of java. It's a read only class and has only getter methods.
- The Object class is beneficial if you want to refer to any object whose type you do not know for whatever reason.
    - The parent class reference variable can refer to the child class object (known as upcasting).
- The Object class provides some common behaviors to all objects such as an object can be compared, can be cloned or can be notified, etc.



Methods of the object class



Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Object cloning

- **Object cloning** is a way to create exact copy of an object. The clone() method of the Object class is used to clone an object.
- The **java.lang.Cloneable interface** must be implemented by the class whose object should be cloneable.
  - If not implement the method generates **CloneNotSupportedException**.
- Advantages: saves line of code, easiest and most efficient way to copy objects, fastest way to copy arrays.
- Disadvantages: need to implement the interface, does not have a constructor…
- Example: Student with cloning (com.pgr103.cloning)

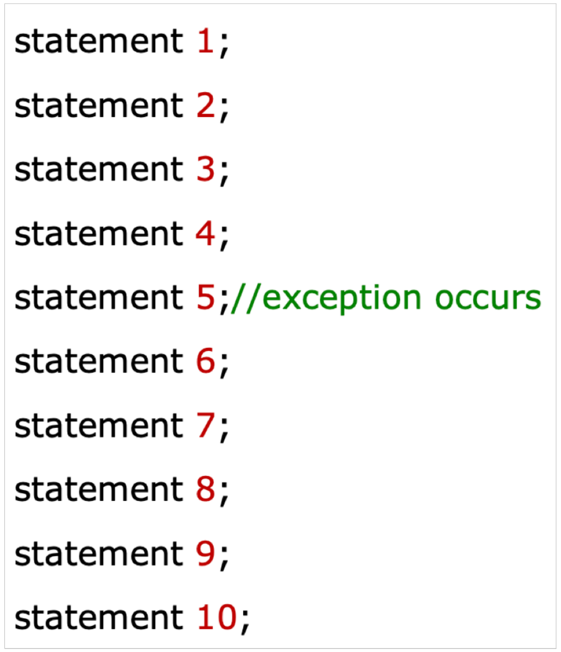
Exception handling

- **Exception Handling in Java** is one of the powerful *mechanisms to handle the runtime errors* so that normal flow of the program can be maintained.
- What is an exception?
  - Exception is an **abnormal condition/state in your program!**
- In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Why do we need exception handling?

- The core advantage of exception handling is **to maintain the normal flow of the application**.
  - An exception normally disrupts the normal flow of the application that is why we use exception handling.

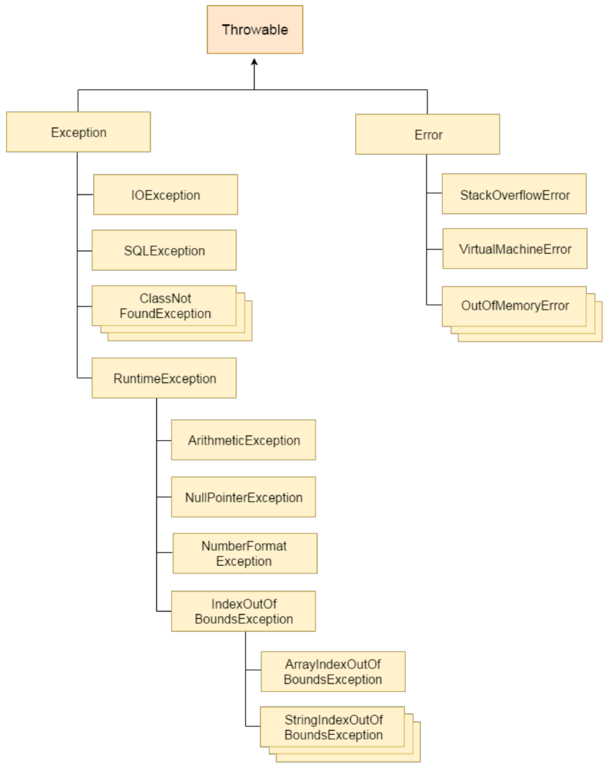
  - Suppose there are **10 statements** in your program and there occurs an **exception at statement 5**. Thus, the rest of the code will not be executed i.e. statement **6 to 10** will **not** be **executed**. If we perform **exception handling**, the **rest** of the statement will be **executed**. That is why we use exception handling in Java.



```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Hierarchy of Java Exception classes

- The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error.



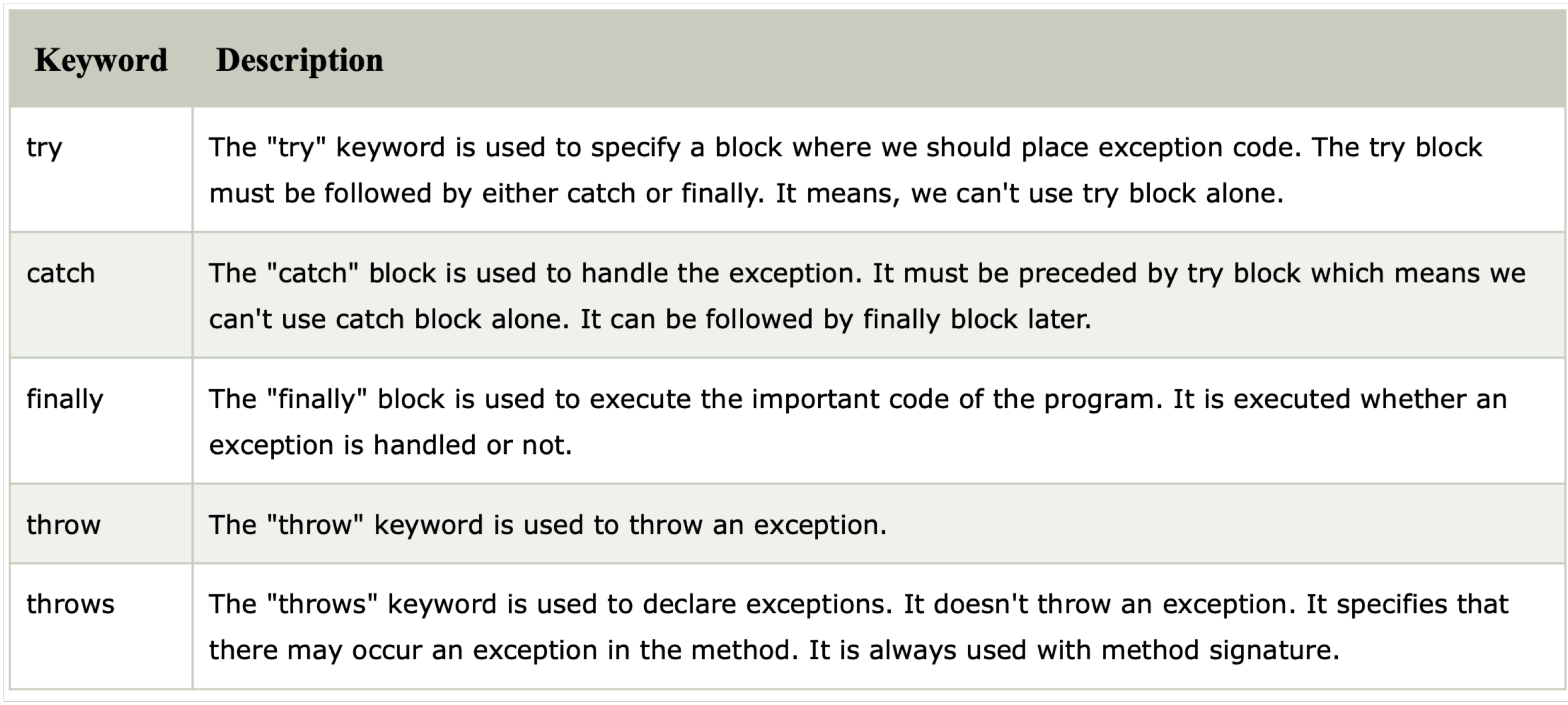
Types of exceptions

- There are mainly two types of exceptions: checked and unchecked.
  - An error is considered as the unchecked exception.
- According to Oracle, there are three types of exceptions:
  - Checked Exception
  - Unchecked Exception
  - Error

Difference between Checked and Unchecked Exceptions

- **1) Checked Exception**
- The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions
  - e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.
- **2) Unchecked Exception**
- The classes which inherit RuntimeException are known as unchecked exceptions
  - e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
- **3) Error**
- Error is irrecoverable
  - e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Exception key words



Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Exception handling example

- Java Exception Handling where we using a try-catch statement to handle the exception (com.pgr103.exceptionexamples)

Common scenarios of exceptions (i)

- There are given some scenarios where unchecked exceptions may occur.
- **1) A scenario where ArithmeticException occurs**
- If we divide any number by zero, there occurs an ArithmeticException.
    - **int** a=50/0; //ArithmeticException

- **2) A scenario where NullPointerException occurs**
- If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.
    - String s=**null**; System.out.println(s.length()); //NullPointerException

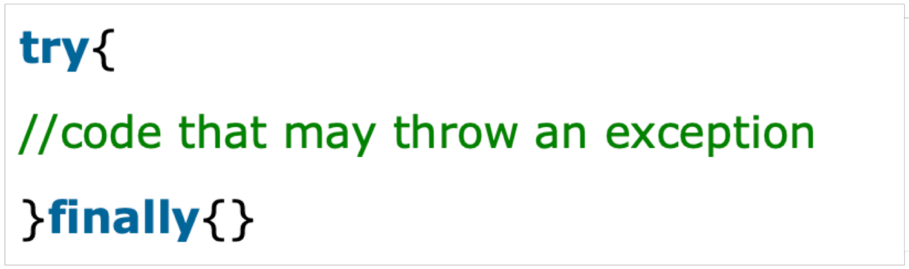
Common scenarios of exceptions (ii)

- **3) A scenario where NumberFormatException occurs**
- The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.
  - String s="abc"; **int** i=Integer.parseInt(s);//NumberFormatException

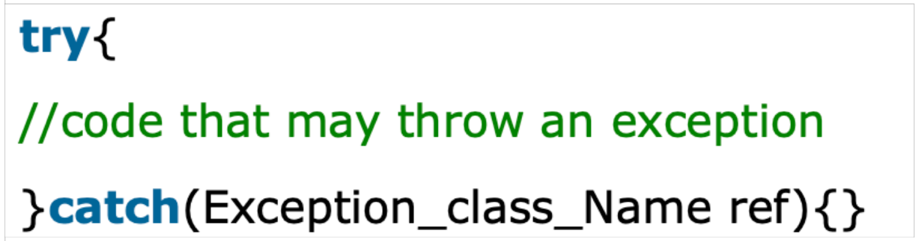
- **4) A scenario where ArrayIndexOutOfBoundsException occurs**
- If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:
  - **int** a[]=**new int**[5]; a[10]=50; //ArrayIndexOutOfBoundsException

Java try-catch block

- The **try** block is used to enclose the code that might throw an exception. It must be used within the method.
- If an exception occurs at the particular statement of the try block, the rest of the block code will not execute.
    - It is recommended not to keeping code in the try block that will not throw an exception.
- The try block must be followed by either catch or finally block.



```
try{
//code that may throw an exception
}finally{}
```



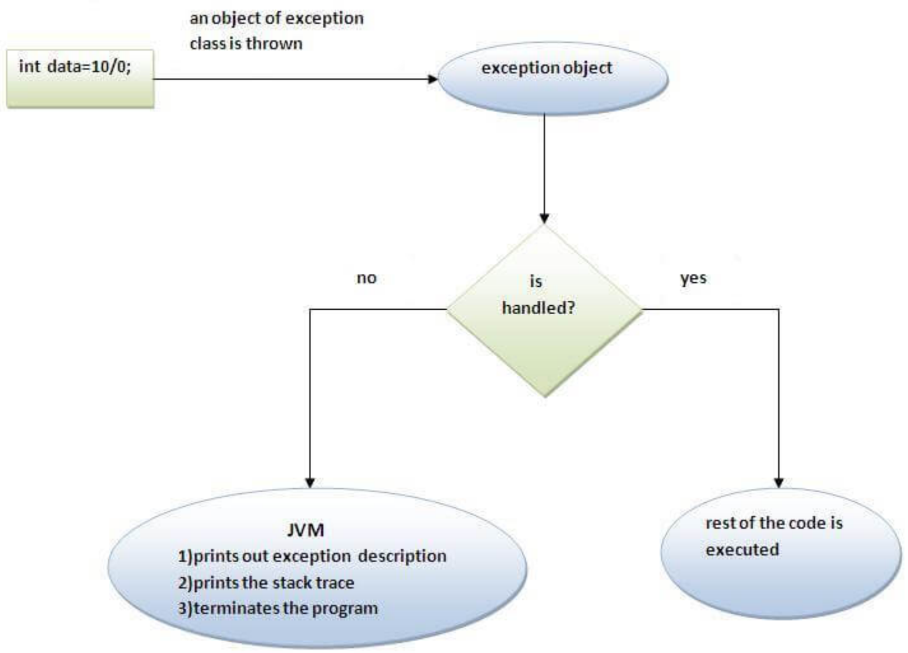
```
try{
//code that may throw an exception
}catch(Exception_class_Name ref){}
```

Try catch block

- **Java catch block is used to handle the Exception by declaring the type of exception within the parameter.**
- The **declared exception must be the parent class exception** (i.e., Exception) **or the generated exception** type.
- The **catch block must be used after the try block only**. You **can use multiple catch blocks with a single try block**.
- Example 1: Not try catch (TryCatchExample1)
- Example 2: With try catch (TryCatchExample2)
- Example 3: Try catch no exception thrown (TryCatchExample3)
- Example 4: We handle the exception using the parent class exception (TryCatchExample4)
- Example 5: Try catch with custom message (TryCatchExample5).
- Example 6: Try and catch for file (TryCatchExample6)

Common scenarios of exceptions (ii)

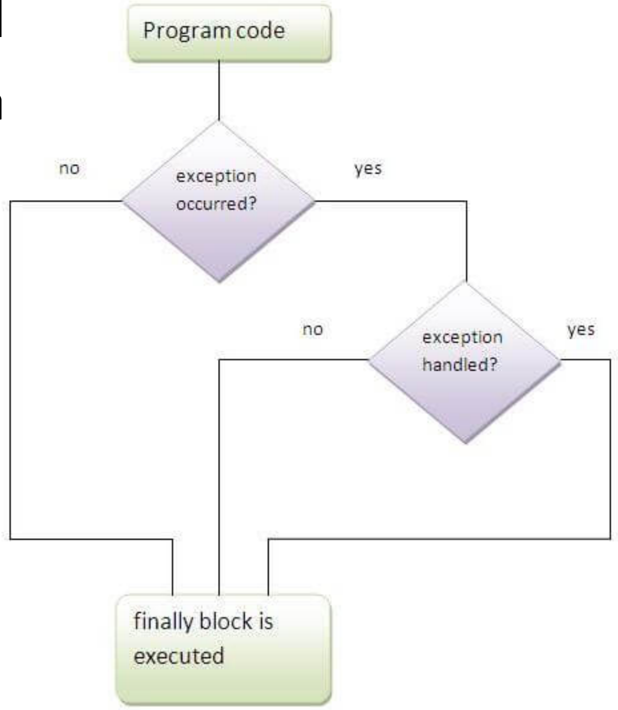
- The JVM **first checks whether the exception is handled or not.** If exception is not handled, JVM provides a default exception handler that performs the following tasks:
  - Prints out exception description.
  - Prints the stack trace (Hierarchy of methods where the exception occurred).
  - Causes the program to terminat
- But if the exception is handled normal flow of the application is executed.



an object of exception class is thrown
int data=10/0;
exception object
is handled?
no
yes
JVM
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

Finally block

- **The finally block** is a block that is used *to execute important code* such as closing connection, stream etc.
- The finally block is always executed whether exception is handled or not.
- Java finally block follows the try or catch blocl
- A finally block in java can be used to put "clea closing a file, closing connection, etc.



Finally block examples

- Example 1: A finally example where **exception does not occur (TestFinallyBlock)**.
- Example 2: finally example where **exception occurs and not handled (TestFinallyBlock1)**.
- Example 3: Finally example where **exception occurs and handled (TestFinallyBlock3)**.
- **Hint: For each try block there can be zero or more catch blocks, but only one finally block.**
- **Important note: The finally block will not be executed if program exits (either by calling System.exit() or by causing a fatal error that causes the process to abort).**

Throw keyword

- The Java throw keyword is used to **explicitly throw an exception**.
- We can throw either checked or unchecked exceptions in java by using the throw keyword.
- The throw keyword is **mainly used to throw custom exceptions**.
- Example: We have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote (TestThrow1).

Where are we now?

- Constructors
- Static key word
- This key word
- Inheritance
- Aggregation
- Method overloading and overriding in context with OOP
- Polymorphism
- Abstract classes, Interfaces
- Encapsulation, object class, cloning, exception handling
- Next time, repetition!