Exceptions are also objects, but they are special
i.e. They are throwable and catchable

```java
try {
    anobj.f()  // f may throw an exception
}
catch (Exception e) {
}
finally {
}
```
Creating New Exception Types

Class MyException extends Exception {
    // a new exception type
}

class MyClass {
    public void f(int i) throws MyException {
    }
}

This creates a checked exception which can be thrown by other classes.

Catch block has to be specified when a member function which can throw a checked exception is invoked.
Some exceptions are unchecked

Subclasses of

RuntimeException
  Array index out of bound exception
  Null pointer exception
  Arithmetic exception
  Illegal monitor state (thread wanting to wait on a monitor object of which it is not the owner) etc

Error
  Assertion error
  Linkage error
  Virtual machine error etc
Try block is executed till an exception gets thrown; if not, the block completes.

There could be multiple catch clauses.

The first matching (type based) catch is selected for execution.

Finally clause is optional.

Finally clause is executed always if present irrespective of how try terminated (break/exception/normal).
Break and Labeled Break

break; exits from any block  
  e.g. Exit from switch, for, while, do blocks
example: for (...) { ...; ... break; ....}

Unlabeled break terminates the innermost block statement

To break out of an outer statement labeled break is used

alabel : ...
   for (i=....) {
      for (j = ... ) { break alabel;}
   }

break is not the same as GOTO statement!
Continue and Labeled Continue

Continue

  skips to the end of current loop's body (while/do/for)
  loop termination is evaluated
  loop may continue with next iteration
  for (...) { if (..) continue; ...}

Continue can be used to skip the rest of the body over trivial cases

To skip the current iteration of an outer loop, labeled continue is used

alabel : for (i=....)
          for (j=....) { .... ; continue alabel; ....}
assert expression;
    if the expression evaluates to true, throw an error
    AssertionError
assert exp1 : exp2;
    value exp2 is sent to AssertionError's constructor

See the demo programs for instructions on compilation and execution of Java code with assertion facility. In old Java compilers, assertion is not a keyword and it has been added later.
Use of Assertions in Software Systems

A Boolean expression placed in a program where its evaluation is always true.

Typically supported as text annotations or embedded executables.

Focus is on what part rather than how part of the system.

 Detection, classification and Diagnosis of errors.
Insert (value: T)

Before execution, assert:

\[ \text{Count} < \text{capacity} \]

-------Code for insert -------

After execution, assert:

\[ \text{Count} = \text{old count} + 1 \]
\[ \text{Count} \leq \text{capacity} \]
\[ \text{Values}[\text{old count}] = \text{value} \]
Assertions in Practice

Contract view
   Needs to be enforced by following it as a contract
   A good design process

Defensive programming view
   An assertion expresses programmer’s intentions
   Failure? – handle exception/abort
   A good debugging process
The contract view

Example: Meyer’s *design by contract* method

Express contracts

Assign the responsibilities

  ad-hoc redundant checks are not needed

Produce contract documentation based on assertions
The contract

Parties involved: client (caller class) and server (callee class)

Preconditions --- the server's business logic benefits from it since a message is not accepted if precondition is not satisfied. Precondition is an obligation for the client.

Postconditions – the client's code benefits from postconditions of member functions defined in the server. Since postcondition is checked by the server, the caller need not again check the validity of the return results. I

If preconditions or postconditions are not satisfied, assertion errors or exceptions can be generated.
#include <assert.h>

....

void insert (int i) {

    assert (count < CAPACITY);

    ....

}

main () {

    ... insert (element); ...

}
Preconditions
   To be asserted before method execution begins

Postconditions
   To be asserted after method execution before returning the result

Class Invariants
   To be asserted
      after every object creation
      after every method execution
      i.e. in observable states only,
      not necessarily during method execution
An Example: design by contract in Eiffel
-- Use assertions in Java

insert (value: T) is

require
  count < capacity

do
  -- Actual functional code

ensure
  count = old count + 1
  count <= capacity
  values[old count] = value

end