Lecture-5

- Misc Topics
  - finalize
- Exceptions
  - Finally
- Generics

Misc topics - finalize

finalize

- It is a way to clean up resources when an object is going out of scope.
- Java has no explicit destructor as in C++
- finalize
  - A function that can be defined in a class.
  - Usually has resource (open file descriptors, open sockets, etc.) clean up methods.
  - Is the closest that comes to a destructor.
  - Is called when Java garbage collector runs.
- Java garbage collector
  - A daemon that runs to clean up resources.
  - Calls finalize methods of objects that are about to cleaned up.

Exceptions

Exceptions

- A way to handle error or unexpected conditions.
- Used to ensure that error conditions are handled gracefully.
- In Java, there is a class called Exception that is used to handle any generic exceptional condition.
  - Exception is derived from Throwable
- Many kinds of specific exceptions are also available
  - I/O exceptions
  - Array out of bound exceptions
  - Class not found exception
  - No such method exception
- Users can define their own exceptions
  - Derive their class from Exception

How to catch exceptions

- Use try catch statements

```
try
  {
    // some code
   }
catch (AnyException e)
  {
    // Error handling code
  }
```

- AnyException is any Exception
  - Can be Java defined exception, or user defined

Throwing Exceptions

- Throw an exception using

  throw (e);

- Any exception – user defined or Java defined exception (e) can be thrown.

Throwing exceptions … contd.

- The function that throws any exception should define it.

```
void function1( ) throws AnyException
    {
        // code
                // in case of error conditions
        throw (new AnyException( ) );
    }
```

- Any function calling myFunction should catch the exception

```
void function2 ( )
    {
        try
            {
                function1( )
            }
        catch (AnyException e) // Or parent of AnyException
            {
                // Error handling code
            }
    }
```

Java – finally

- finally – a way to handle any left over (cleanup) issues.
- Should be present in the end, after try and catch are done.
- Typically used to clean up resources, open files, file descriptors, sockets, etc.

Generics

Generics – motivation

- Consider the following classes to set/get values to a variable.

```
class intClass
{
    private Integer intval;
    public void setValue (Integer i) { intval = i; }
    public Integer getValue ( ) { return (intval); }
}
class doubleClass
{
    private Double dblval;
    public void setValue (Double d) { dblval = d; }
    public Double getValue ( ) { return (dblval); }
}
```

- Same code, but two classes are used
  - One for each data type

Generics motivation … cont.

- Problem here
  - One class is needed for each data type.
  - But the code itself is almost the same.
- Generics are used to solve this issue.
- Generics
  - Generic classes that can be used with class
  - Same code can be used with any class.
  - Avoids code repetition.
  - Similar to C++ template classes

Generics

- The problem above is that the same code is repeated for different types
- Solution: Generic types
  - Uses a "generic" (any) type as a parameter.
  - Objects of any specific non-basic data types can be instantiated. E.g.

```
// Generic class declaration
 public class GenericClass<T>
  {
    // code
 }

// Instantiating two object of type Integer and Double
GenericClass<Integer> gInt = new GenericClass(Integer)( );
GenericClass<Double> gDbl  = new GenericClass<Double>( );
```

Generics with multiple generic types

- Use different types in the class definition.
- E.g. for two generic types

```
interface Pair <k, v>
 {
    public k getKey( );
    public v getValue( );
 }

class OrderedPair<k, v> implements Pair<K, V>
 {
        ..
 }
```