Object Oriented Programming CS F213 Amit Dua

BITS Pilani
Pilani Campus

Slides Taken from the slides prepared by Dr. Jennifer

BITS Pilani
Pilani Campus

Java Object Model

Topics for today

- Analysis of objects and their capabilities: Reflection
- Examples
- Enumeration
- Garbage collector

Properties

- Class
- superclass
- Interface
- Package
- Names and types of all fields
- Parameter types of all constructors
- Name, parameter types and return types of all methods
- Invoking the methods

Class and parentclass

```
child c1 = new child(5);
    Class cl1= c1.getClass();
    Class cl2 = cl1.getSuperclass();
    System.out.println("class name= "+cl1.getName());
    System.out.println("Parent class name=
    "+cl2.getName());
```

Inerfaces

```
Class[] interfaces = cl1.getInterfaces();
    for(int i =0; i<interfaces.length;i++){
        System.out.println("Interface "+i+ "= "+
    interfaces[i].getName());
```

Package

```
Package pkg = cl1.getPackage();
    System.out.println("Package= "+pkg.getName());
```

https://docs.oracle.com/javase/7/docs/api/java/lang/Package.html

Fields

```
Field[] fields = cl1.getDeclaredFields();
for(int i=0; i<fields.length;i++){
		if(Modifier.isPublic(fields[i].getModifiers()))
		   System.out.println(fields[i].getName());
```

Constructors

```
Constructor[] cons = cl1.getDeclaredConstructors();
for(int i = 0; i<cons.length;i++){
        Class[] params = cons[i].getParameterTypes();
        System.out.print("child(");
        for(int j=0;j<params.length;j++){
            if(j>0)System.out.print(", ");
            System.out.print(params[j].getName());
        }
        System.out.println(")");
```

Methods

```
Method[] m1 = cl1.getDeclaredMethods();
    for(int i=0;i<m1.length;i++){
        Class[] params = m1[i].getParameterTypes();
        System.out.print(m1[i].getReturnType()+" " +
    m1[i].getName()+"(");
        for(int j=0;j<params.length;j++){
            if(j>0)System.out.print(", ");
            System.out.print(params[j].getName());
        }
        System.out.println(")");
```

Invoking methods

```
try{
        child c3 = new child(6);
        System.out.print(m1[2].getName()+ " ");
    int p = (int) m1[2].invoke(c3);
    System.out.println(p);
        System.out.print(m1[1].getName() + " ");
    System.out.println(m1[1].invoke(new child(5)));
        System.out.print(m1[3].getName());
        System.out.print(" Before value of j= "+c3.getJ());
        Object o = m1[3].invoke(c3, new Object[]{10});
        System.out.print(" value of j= "+c3.getJ());
    }
    catch(IllegalAccessException e){}
    catch(InvocationTargetException e1){}
}
```

Demo

ReflectionClass2.java

Enum Type

- Used for representing a group of named constants in programming
- Enum in java is more powerful than C/C++
- In Java, we can add variables, methods and constructors to it.
- Enum can be declared outside the class or inside the class but not inside the method.

```
class Test{
enum Color{
 RED, GREEN, BLUE;
 }
 public static void main(String[] args) {
    }
}
```

Features of enum

- Enum is internally implemented using class

```
/* internally above enum Color is converted to
class Color {
public static final Color RED = new Color();
public static final Color BLUE = new Color();
public static final Color GREEN = new Color(); }*/
```

- Constants represents an object of type enum
- Constants are always implicitly public static final
  - It can be accessed using enum name
  - Child enums can not be created.
- It can be passed as an argument to switch statements

Features of enum

- All enums implicitly extend java.lang.Enum class
- toString() returns the enum constant name
- values() method can be used to return all values present inside enum
- ordinal() method is used to retrieve the constant index
- Enum can contain constructor and it is executed separately for each enum constant at the time of class loading.
- We cant create enum objects explicitly and hence we cannot invoke the enum constructor directly
- Enum can contain concrete method and not abstract methods.

Enum Example

Enumex.java

innovate achieve lead

Garbage collector

- finalize()
  - This method is called before garbage collection when an object has no more references.
  - It could be overridden to dispose system resources, perform clean up and minimize memory leaks.
  - finalize() method is called just once on an object
  - protected void finalize()
- gc()
  - It is used to invoke the garbage collector to perform clean up
  - It is found in System and Runtime classes.
  - public static void gc()

Java Runtime class

- It is used to interact with the Java runtime environment
- It provides methods to execute a process, invoke GC, get total and free memory etc.
- Only one instance of the java.lang.Runtime class is available for one Java application

Garbage Collector : gc()

GarbageCollector.java

Finalize()

- The *finalize()* method called by Garbage Collector not JVM. Although Garbage Collector is one of the module of JVM.
- Object class *finalize()* method has empty implementation, thus it is recommended to override *finalize()* method to dispose of system resources or to perform other cleanup.
- The *finalize()* method is never invoked more than once for any given object.
- If an uncaught exception is thrown by the *finalize()* method, the exception is ignored and finalization of that object terminates.