What is abstraction?

Hiding the internal details and just highlight the set of service that we are offering.

Advantages:

Security

Enhancement will become easy

Easiness for end-user

Maintainence is easy without interfering end user.

By using interface and abstract we can achieve Abstraction.

What is Constructor and why it is used?

Whenever we create an object constructor will be executed automatically to perform some intialization.

What is the need for instance block?

Other than intialization, if we want to perform any activity for evvery object creation.

What is difference between constructor and instance block?

Both constructor and instance block have their own different purposes and and replacing one concept with another concept may not work always.

Both constructor and instance block will be executed for every object creation but instance block first followed by constructor next.

**static int** *count* = 20;

{

*count*++;

}

What are rules to write constructor?

Name of constructor should match the class name

Return type constructor should not applicable for constructor

By mistake if we trying to declare return type for constructor then we wont get compile time error, because compiler would treat it as method.

What are the modifiers applicable for constructor?

private, protected, public and default. If we are trying to use any other modifier we will get compile error.

What is default constructor?

Compiler is responsible to generate default constructor.

If we are not writing any constructor then only compiler will generate default constructor, i.e. if we writing atleast one constructor compiler won’t generate default constructor.

Prototype of default constructor

1. It is always no-arg constructor

2. The access modifier of default constructor is exactly same as class(This rule is applicable for public and default)

3. It contains only one line

super(); //Is a no argument call to super class constructor.

Few examples with for compiler generated constructor againist programmer generated code.

|  |  |
| --- | --- |
| Programmer’s | Compiler Generated Code |
| class Test {} | class Test{Test() {super();}} |
| public class Test {} | public class Test() {public Test() {super();}} |
| public class Test {void test() {}} | public class Test{public Test() {super();} void test() {}} |
| class Test{Test() {}} | class Test {Test(){super();}} |
| class Test {Test(int i) {super();}} | class Test { Test(int i) {super();}} |
| class Test{Test(int i) {this(10);} Test(int i){} } | class Test{Test(int i) {this(10);} Test(int i){super();} } |

What are some rules to define super() and this() constructor calls?

1. The call to the super() or this() should be on the first line of the constructor

2. We can take either super() or this() but not both

3. We can use super() or this() only inside constructor, if we are using outside of constructor we will get compile time error.

Difference b/w super(), this() and super, this

|  |  |
| --- | --- |
| super(),this() | super.this |
| These are constructor calls | These are keywords to refer super and current class instance members |
| Only inside constructor | Everywhere expect static area |
| We can use only once in constructor | We can use any number of times |

What is Coupling and Cohesion?

The **degree of dependency** between the components is called coupling. Tightly coupling is not good programming practice because it has several serious dis-advantages.

Without affecting remaining components we can’t modify any component and hence

1. Enhancement will become difficult

2. It suppress re-usability

3. It reduces maintainability of the application

Cohesion:

For every component a clear well defined functionality is defined.