**What is Coupling in Java?**

Coupling is nothing but the dependency of one class on the other. If one object in a code uses the other object in the program, it is called **coupling**.

In coupling, two classes or objects collaborate and work with each other to complete a pre-defined task. It simply means that one element requires another element to complete a function. It is known as **collaboration** when one class calls the logic of the other class.

There are two types of coupling

Tight Coupling

Loosely Coupling

***Loosely coupling means reducing the dependencies of a class that uses the different classes directly. Tight coupling means classes and objects are dependent on one another.***

**Tight Coupling**

When two classes are highly dependent on each other, it is called tight coupling. It occurs when a class takes too many responsibilities or where a change in one class requires changes in the other class.

In tight coupling, an object (parent object) creates another object (child object) for its usage. If the parent object knows more about how the child object was implemented, we can say that the parent and child object are tightly coupled.

Example:-

**class** SmartWorker {

**public** **void** work() {

System.***out***.println("smart worker working");

}

}

**class** LazyWorker {

**public** **void** work() {

System.***out***.println("Lazy worker working");

}

}

**class** Manager {

SmartWorker sw;

LazyWorker lw;

**public** Manager(SmartWorker sw, LazyWorker lw) {

**this**.sw = sw;

**this**.lw = lw;

}

**public** **void** ManageWork() {

sw.work();

lw.work();

}

}

**public** **class** TightCoupling {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

SmartWorker sw = **new** SmartWorker();

LazyWorker lw = **new** LazyWorker();

Manager mn = **new** Manager(sw, lw);

mn.ManageWork();

}

}

**class** ExtraordinaryWorker {

**public** **void** work() {

System.***out***.println("ExtraOrdinary worker working");

}

}

**Loose Coupling**

When two classes, modules, or components have low dependencies on each other, it is called loose coupling. Loose coupling means that the classes are independent of each other. The only knowledge one class has about the other class is what the other class has exposed through its **interfaces** in loose coupling. If a situation requires objects to be used from outside, it is termed as a loose coupling situation.

Here, the parent object is rarely using the object, and the object can be easily changed from external sources. The loose coupling in Java has the edge over tight coupling as it reduces code maintenance and efforts. A change in one class does not require changes in the other class, and two classes can work independently.

**interface** IWorker{

**void** work();

}

**class** Manager1 {

IWorker worker;

**public** Manager1(IWorker worker) {

**this**.worker=worker;

}

**public** **void** ManageWork() {

**this**.worker.work();

}

}

**class** SmartWorker1 **implements** IWorker{

**public** **void** work() {

System.***out***.println("smart worker working");

}

}

**class** LazyWorker1 **implements** IWorker {

**public** **void** work() {

System.***out***.println("Lazy worker working");

}

}

**class** ExtraordinaryWorker1 **implements** IWorker{

**public** **void** work() {

System.***out***.println("ExtraOrdinary worker working");

}

}

**public** **class** LooslyCoupling {

**public** **static** **void** main(String[] args) {

SmartWorker1 sw = **new** SmartWorker1();

Manager1 mn = **new** Manager1(sw);

mn.ManageWork();

LazyWorker1 lw = **new** LazyWorker1();

Manager1 mn2 = **new** Manager1(lw);

mn2.ManageWork();

ExtraordinaryWorker1 ew = **new** ExtraordinaryWorker1();

Manager1 mn3 = **new** Manager1(ew);

mn3.ManageWork();

}

}

**Differences Between Loose Coupling and Tight Coupling**

The following table lists the differences between loose coupling and tight coupling.

| **Loose Coupling** | **Tight Coupling** |
| --- | --- |
| Objects are independent of each other. | One object is dependent on the other object to complete a task. |
| Better testability | Testability is not as great as the loose coupling in Java. |
| Asynchronous communication | Synchronous communication |

**loose coupling in Java** is much better as compared to tight coupling. It provides better flexibility and reusability of code. As the two classes are independent of each other, it makes changes in the code very easy. It also provides better testability.