

## (1)

(الف)

به میزان ارتباط دو کلاس یا بخش از برنامه میزان **coupling** گفته می شود. هر چه قدر این میزان بیشتر باشد **tightly coupling** اتفاق می افتد و هر چه وابستگی بخش های مختلف به هم کمتر باشد **loosely coupling** اتفاق می افتد هدف ما در طراحی همواره این است که به **loosely coupling** برسیم .

**Cohesion** که میزان انسجام یک مفهوم در برنامه اشاره دارد. در **class cohesion** به این موضوع اشاره میکنیم که هر کلاس تنها باید نماینده یک مفهوم مستقل باشد و نباید به چند مفهوم به صورت همزمان اشاره کند. در **method cohesion** مهم این است که متد ها تنها یک کار را انجام دهند هنگامی که یکی از متد های ما مسئول انجام چند کار مختلف باشد برنامه داری انسجام کمی می باشد . هدف ما در طراحی رسیدن به **high cohesion** می باشد .

**Encapsulation** مفهومی است که در آن سعی میکنیم دسترسی بخش های مختلف برنامه به یکدیگر را تا حد ممکن محدود کنیم به این صورت هر بخش از برنامه هویت مشخص خود را پدا میکند و امکان ایجاد تغییرات ناخواسته تا حد ممکن کاهش پیدا میکند یکی از روش های انجام این کار **private** تعریف کردن **field** های کلاس است به این صورت امکان دسترسی مستقیم کلاس های دیگر به این **field** ها وجود ندارد و کنترل این مقادیر بسیار راحتتر میشود.

**Enum** یک **data type** در می باشد که شمال تعدادی مقادیر ثابت است. به این صورت ما میتوانیم نوع های دلخواه خود را تولید کنیم به عنوان مثلا تولید روز های هفته به وسیله ی **Enum** بهتر از استفاده از **string** ها است چون مقدار های مختلف توسط خود ما محدود می شوند. در جوا مقادیر **Enum** با حروف بزرگ نوشته می شوند. و همگی آنها به صورت پیشفرض **static** و **final** می باشند .

(ب)

در **Factory Method** ما یک **interface** میسازیم و تعدادی **Class** هم آن را **implement** میکنند اما در نهایت این **subclass** ها هستند که تصمیم می گیرند از کدام کلاس باید شی ساخته شود به این صورت که یک کلاس جدا به عنوان کلاس **factory** در نظر میگیریم این کلاس با توجه به شرایط تصمیم می گیرد که از

کدام کلاس باید شی ساخته شود و سپس آن را برمی گرداند این طراحی زمانی به کار می‌آید که interface اصلی ما نمی داند که چه subclass هایی قرار است از آن ارثبری کنند.

هنگامی از Adapter استفاده می کنیم که بخواهیم از یک کلاس که در گذشته ساختیم برای خواسته های جدید client استفاده کنیم . به عنوان مثال فرض کنید با در گذشته یک interface داشتیم که تعدادی کلاس آن را implement میکردند حالا client از ما یک کلاس جدید میخواهد اما این کلاس جدید قابلیت implement کردن interface ما را ندارد پس یک کلاس به نام Adapter می سازیم که interface ما را implement میکند و در آن field ای از نوع کلاس مورد نیاز جدید را نگه داریم میکنیم یک مثال معروف کلاس های wrapper در جاوا هستند که primitive type ها داخل خود نگه داری میکنند و مانند یک adapter عمل میکنند.



