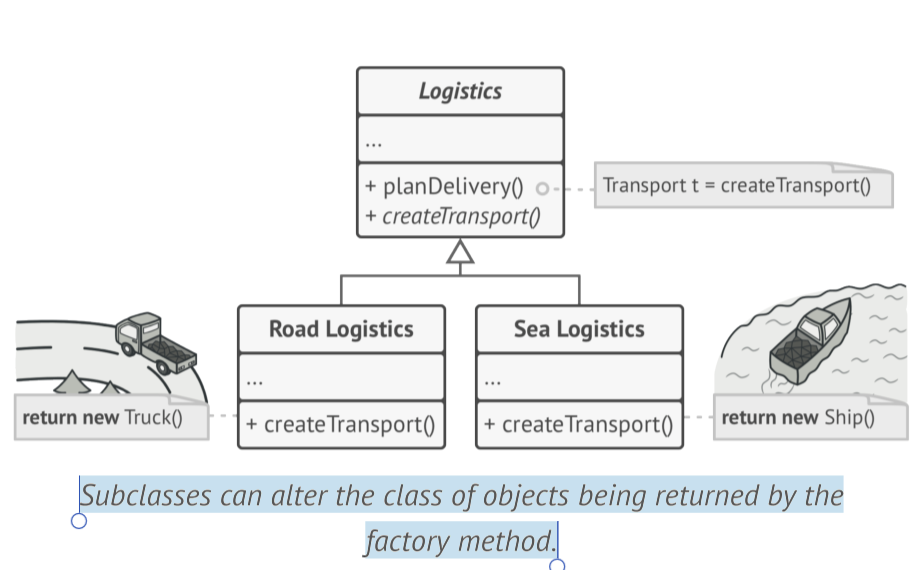
FACTORY METHOD

Also known as: Virtual Constructor

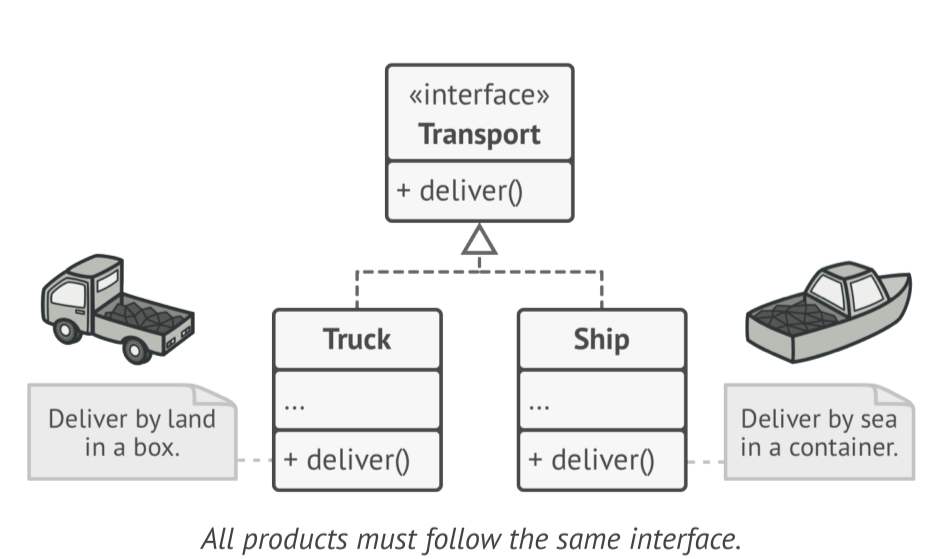
Factory Method is a creational design pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.

The Factory Method pattern suggests that you replace direct object construction calls (using the new operator) with calls to a special factory method.

The objects are still created via the new operator, but it’s being called from within the factory method. Objects returned by a factory method are often referred to as “**products**.”



There’s a slight limitation though: subclasses may return different types of products only if these products have a common base class or interface. Also, the factory method in the base class should have its return type declared as this interface.



UML: [Untitled Diagram.drawio - Page-1 (diagrams.net)](https://viewer.diagrams.net/index.html?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1&title=Untitled%20Diagram.drawio#R7Vxbc5s4FP41ntl9SIeLb3k0jpO2m%2B6midu0TzsyCFsNRizIsZ1fv0cgroIY32LaejrTooOQQN%2Bnc75zwG3pw%2Fnqxkfe7BO1sNPSFGvV0q9amqaqqg7%2FcMtaWPqqElmmPrGELTU8kBcsjHG3BbFwkOvIKHUY8fJGk7ouNlnOhnyfLvPdbOrkZ%2FXQFEuGBxM5svWRWGwWWftaL7W%2Fx2Q6i2dWu5fRmTmKO4snCWbIosuMSR%2B19KFPKYuO5qshdvjqxevy%2BGH96Nw%2BdW8%2Bfg7%2BQ1%2BMv8Z%2Ff72IBrve5pLkEXzssp2Hfnmyr99%2Fbf%2F45t1df14qN4PJxwtxifKMnIVYr6GPEaO%2BeGS2jtcxWJK5g1xoGTZ12YM4o0MbOWTqwrEJt4d9MDxjnxGAYCBOMOqB1ZwRx7pFa7rgDxEwZD7FLWNGffICwyIHTqlggNM%2BE2zSurkeD%2FxKMCtg9XEAfe7ilVET0y0KmOhjUsdBXkAm4Q3zLnPkT4lrUMboPB6ILlwLW6KVQB02mE%2BfEvLw62viIXDjq4FXGTYKfG4wnWPmr6GLOJtQLd5sumgvU%2BaqXWGbZVkbX4jEbpkmYyfT3cPuQu4UFiGdTy%2FM1645HyCSmw45ALyLGDb4MgZZHsJB5lFTU8jOLZjalpgqUdQhIT1nbJ6yqAhdzFUH26yEqXNiWXw4I%2FCQSdzpmDP36kJNLbfhhSHvheVerIse0oghhiKmce44aIKdOxoQRigf34%2F6Gh4lLgsXr2O0OlehxWdD6sIdIxKyCgODl5izuIRvr27mzXxb52HcBHe%2Fmlw5mLfFtFOCqcFjAfdB%2BM6n1sJkf%2FwpAQ2PxzZjGXmdPGxtGTZuonCt7YQ7fgYMwG4JlHnIDFimofKuw8HThtBW0%2FbJ8NTbu23fgwHalQB9YAuLP2XNcKKcw8nhwolaCCd6t6Z7v9wpmqiFaKL3mxtNLs%2FRZJP36damW0OiiaqWgHoOJ7sDevJwomqbtyl2rQFPE6E1cSj35AaYhKtXlah5TfjEdXxx3nGbC%2F85aWALEksxMXYmdDlKDUZogBMxa7Z26wFd%2BCbeDB8EsimuoRr43b4KchmqPnYQI8%2F51PmVKHDHSZzJJ5R8BNAUPT9E9JDiqmyuuu1A0SpIAx0qPMRVjwzxwH3YOAjqZ8ZnKXNAKaMXtcVxpYzeriFles2QMmpZFnXWMuX7uYli5t%2FxI0Pd6%2FFoMnm5mtxot6Ppjwvt9xYzElC1QW6wmpGz499XzcQ7stFyRi1Egd6OckYvDKT1CvdybDnTK5Mz3IHUFTPZAHEWM3uKmW6BDscu83c6P0%2BZX5OV91nNVG3nJqqZclSrKv0WfWAL2%2F5ldcwRsDy5jtG20zHUC5d6D6WSKB719LpFqSlbYlw36haBOadV1ke31YNImCRxjV3%2FpbabhOm0NwxUIWGkgS6KqXVhHGrbAT6KCtJkFSTxNlQVGYewwf1MhLaI%2FL8BgmUaEnlIHQoK6cqlqZISGl6r7xvijSY%2BjRG31Uq%2BEqivOC72ZFNMyiNAVZr9yp%2BlXA%2BG43%2Fuv4PxbjAej%2B7%2FhqMvn25l%2FTpDHg51J%2BaRwcM%2BgZvhcjU03aVtYzkjDD9AoOD9lz7y8prCJiucZF51E9Nq31KJTWE%2FXPRkcZYgnovVr2jB%2FUoP0uJDjDN9nFQdBvKqn0ughazh8Dzp7Po2t7dT0tDQt7mljD1XQDfQrXqfNyFlKL27MpH5a2cMWc%2FZPhasb5k9lN5ef1N0Mc7R5QTRpbfrC7bdoktDX7CVMlaV1eg5vGz2Q%2F1Gh5fSt6bn%2BLI3riePL6ocYCQkm%2FuWDa8I%2B8avBUyj1vd4JDi%2BWmUba9FolRWadit5bQP4EV649S4LUaGzY7WqWyh7aZ1ClKqoVgEr0DrTTey%2B6hsuhjEl9xscOIhGPGw8qvHxanMZvhdb34SFXaVCCjWVhd1TsFCTVdEgCKhJQIsFITCAS0UIzcikjQI)

A diagram of a product

Description automatically generated

**Applicability:**

\* Use the Factory Method when you don’t know beforehand the exact types and dependencies of the objects your code should work with.

The Factory Method separates product construction code from the code that uses the product. Therefore, it’s easier to extend the product construction code independently from the rest of the code.

For example, to add a new product type to the app, you’ll only need to create a new creator subclass and override the factory method in it.

\* Use the Factory Method when you want to provide users of your library or framework with a way to extend its internal components.

A screenshot of a computer program

Description automatically generated