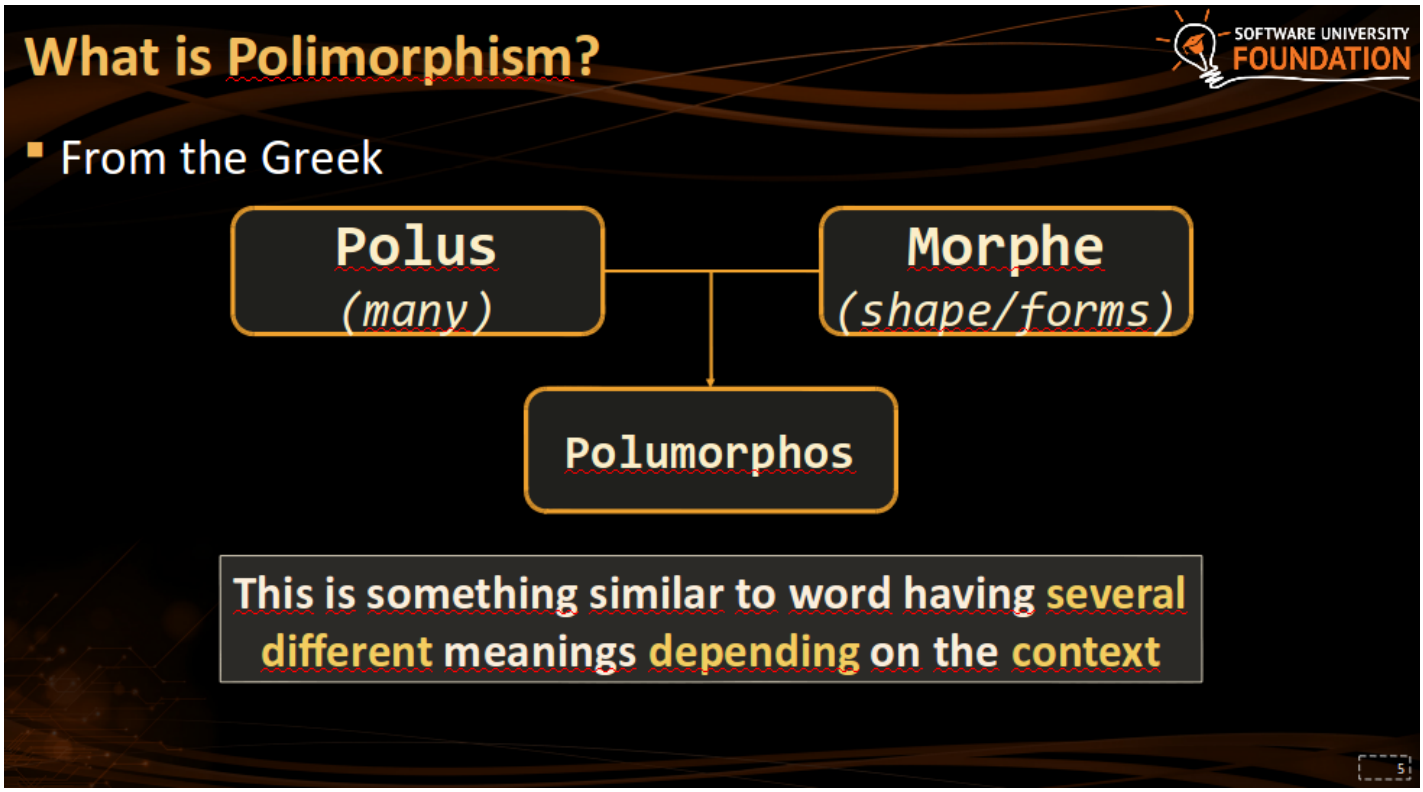
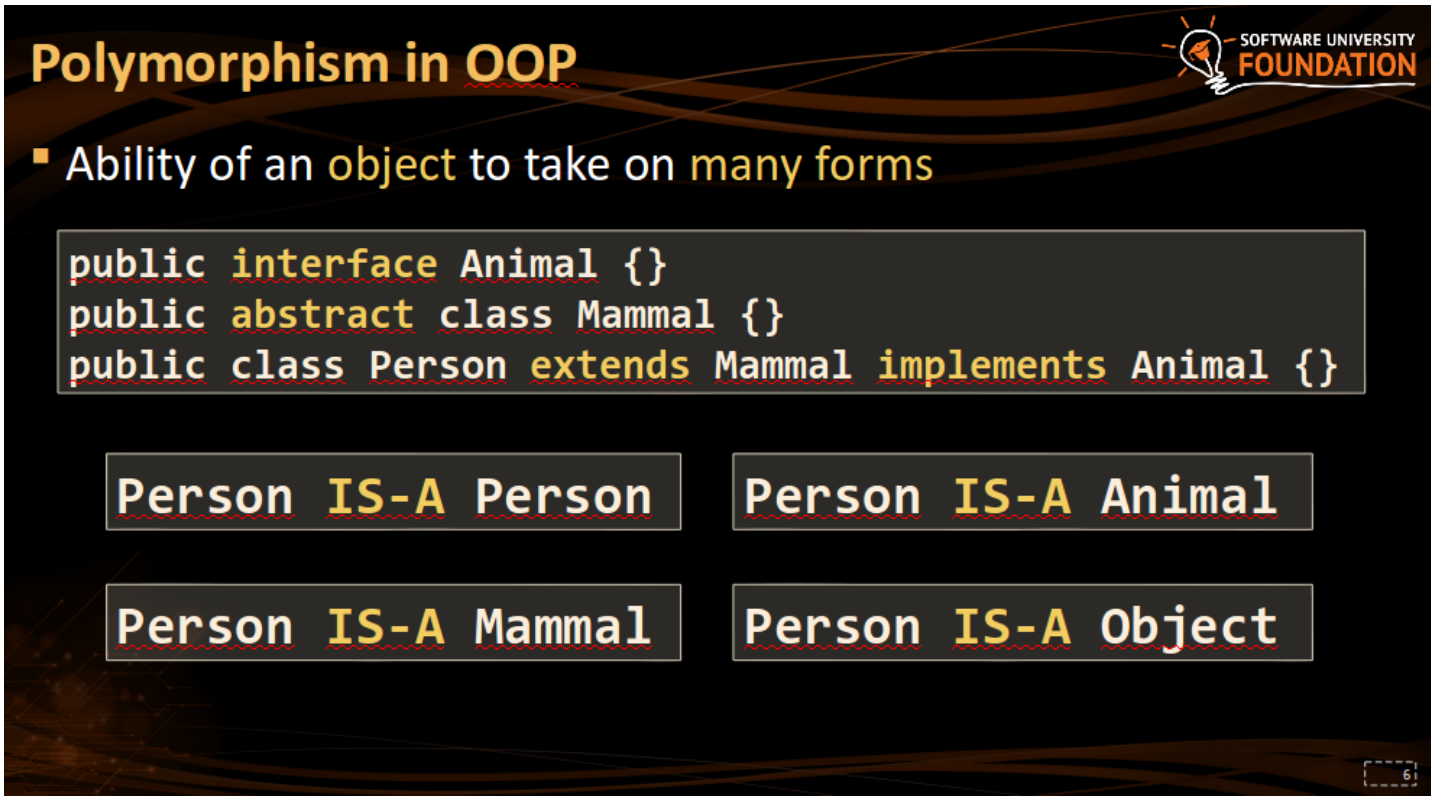
понеделник, 6.03.2017

Polymorphism(Полиморфизъм)



Polymorphism - представлява способността на един обект да променя своята форма.

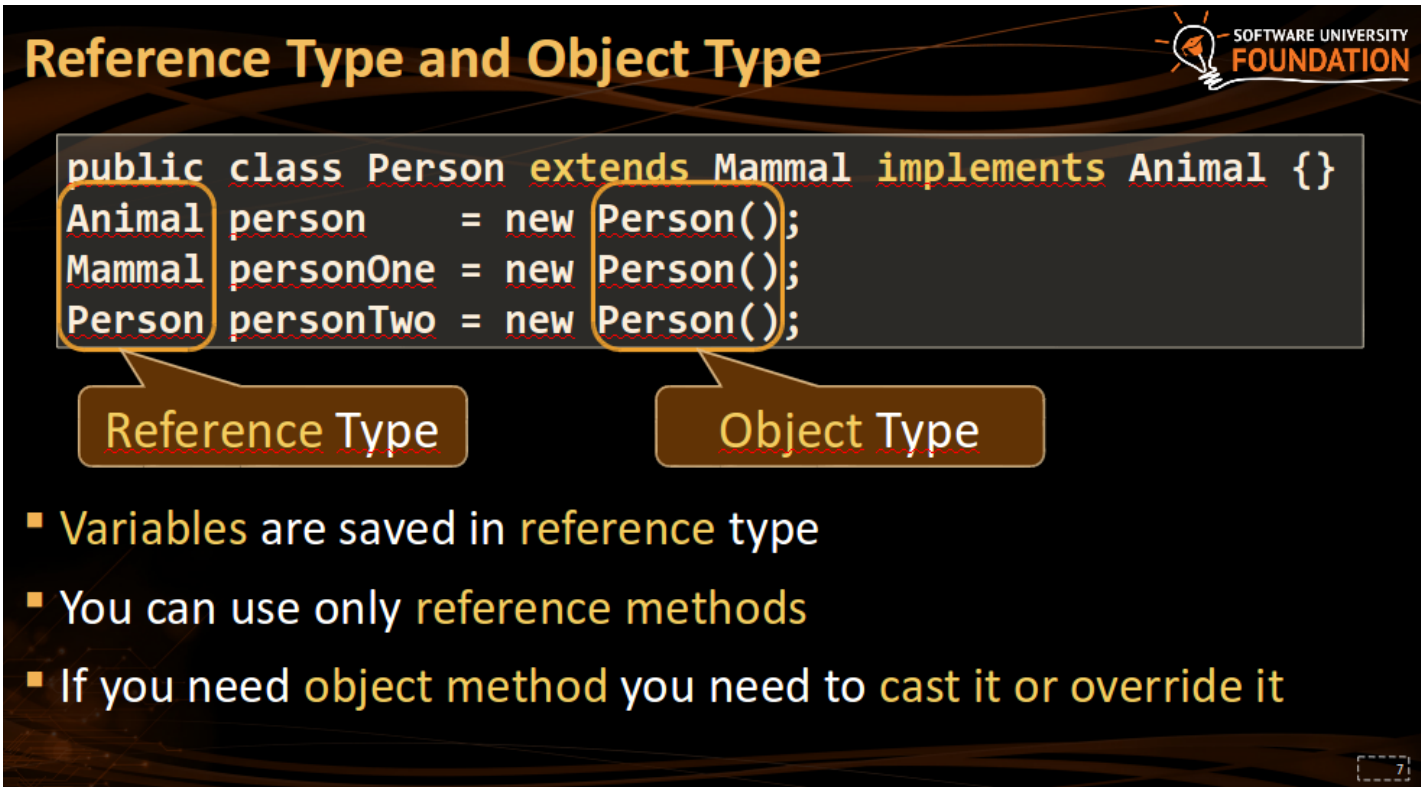
Това е нещо подобно както когато някоя дума и няколко значения, всяко от което се използва в различен контекст.



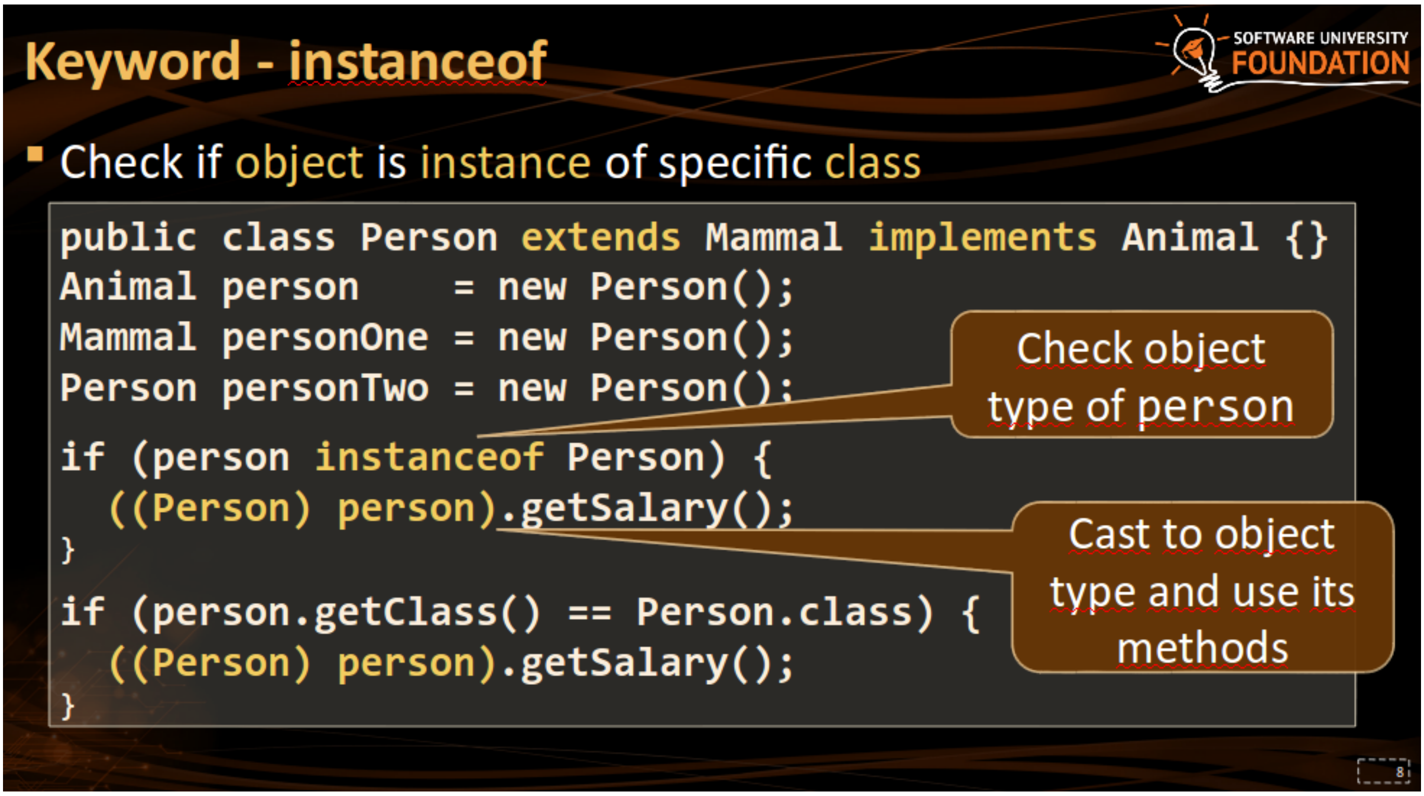
Когато имаме някакво наследяване са много важни връзките **IS-A и колкото повече такива имаме , толкова обектът е по-полиморфен или по-силен полиморфизмът , тоест толкова повече форми може да приема.** За да съществува изобщо полиморфизъм на даден обект, то е достатъчно да има поне 2 такива връзки, а той ги има по подрабиране(по default) , защото всеки обект, който направим(В случая на картинката по-горе Person IS-A Person) , дори да не сме наследили нищо, винаги е

картинката по-горе Person IS-A Person), дори да не сме наследили нищо, винаги е адекватно, както и Person IS-A Object. Тези двете са винаги актуални без значение дали сме направили някакво наследяване или не. За това какъвто и обект да създадем в нашия код, то той е винаги полиморфен, и може да приеме поне 2 форми - неговата си собствена форма , както и тази на Object(В Java всичко е обекти).

Какво се случва, когато създаваме някоя променлива от даден клас ?

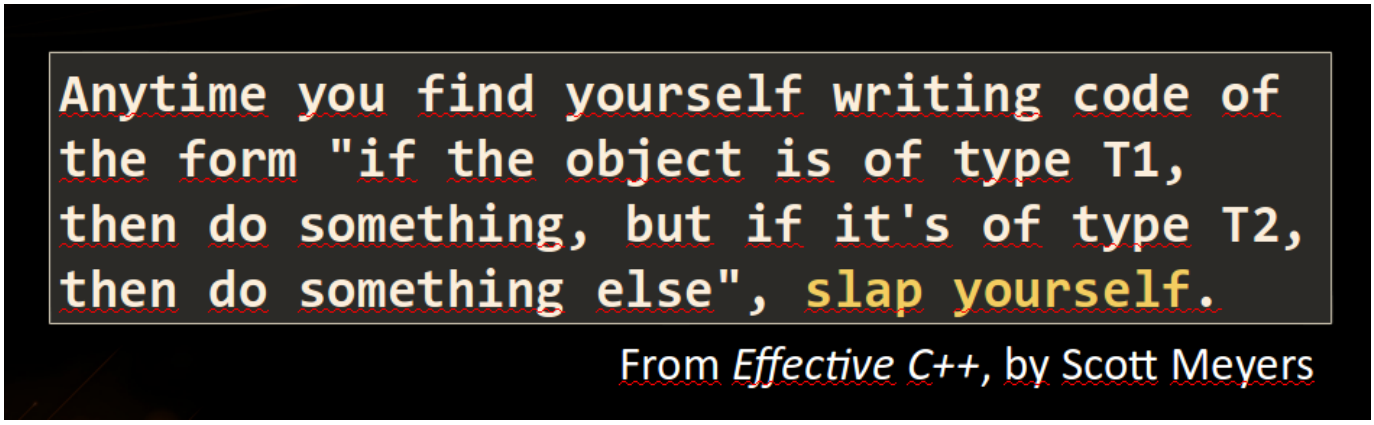


При инициализирането на даден клас вляво ние задаваме каква е референцията, която се пази в рам паммета, а дясната даваме какъв е типът на обекта. **Референцията може да бъде родител(parent) на object-a(обекта). Когато достъпваме методите на нашия обект, ние реално достъпваме методите на референцията, а не конкректно методите на обекта**. Ако се нуждаем от метод на дадения обект, ние трябва да кастнем(cast) тази референция към нашия обект или да презапишем(override-нем) по някакъв начин този обект. Как става това ? Показано е на следващата картинка.



person-a от горния пример е от reference type animal и от object type Person, следователно ако попитаме дали person е инстанция на Person, ще ни върне true. И след това ако

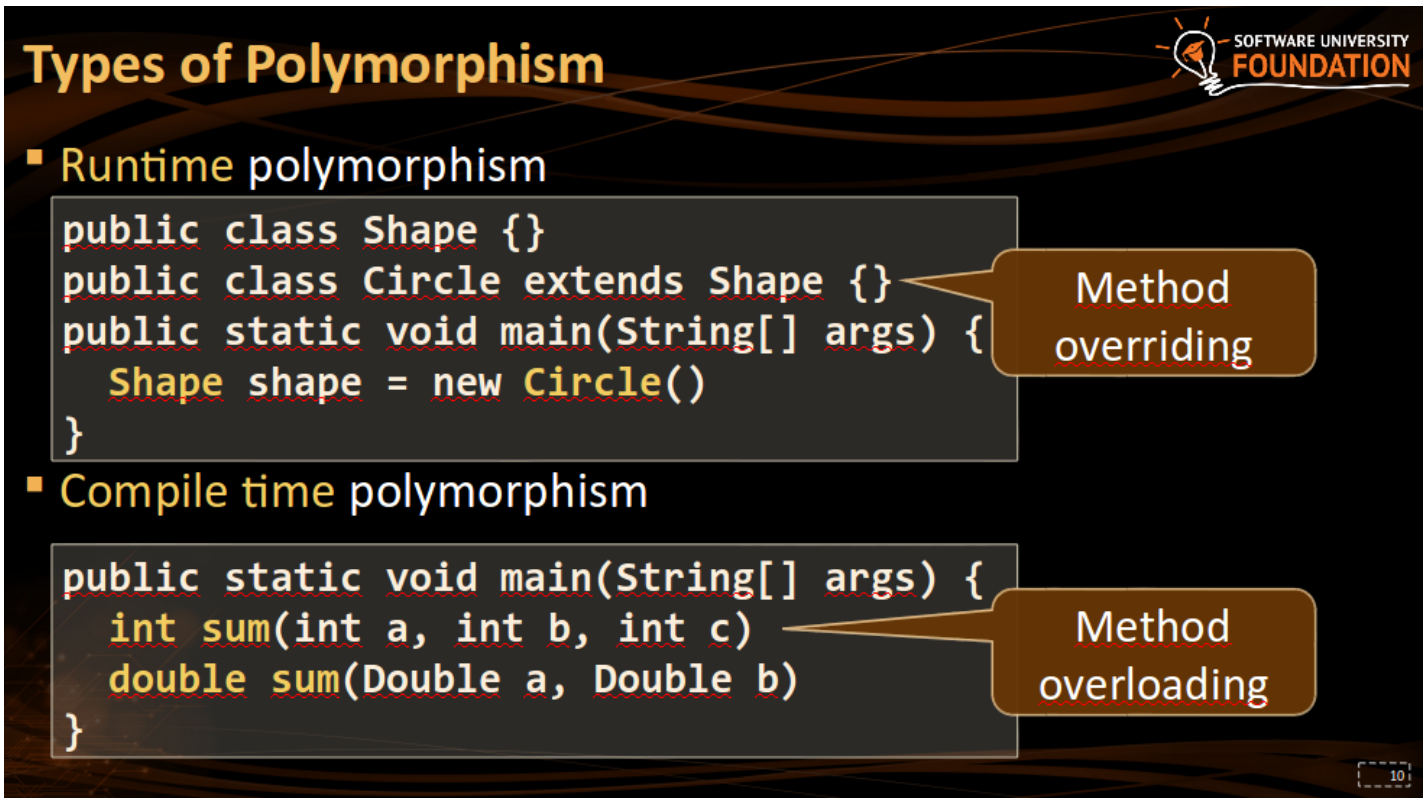
искаме да ползваме някой метод от нашия обект, първо трябва да го кастнем към object type-а му. InstanceOf проверява каква е инстанцията на обекта, а кастването го правим, за да можем да му ползваме методите. Има и друг начин да се направи това - като им сравним класовете. **Дали е добра практика използването на ключовата дума instance of ?**



Скот Майерс(Scott Meyers) е казал следното: "Всеки път когато се усетите , че пишете код от сорта на - ако обектът е от тип 1 направете нещо, ако обектът е от тип 2 направете нещо друго , то тогава си набийте един шамар, изтрийте си кода и измислете нещо по-умно". Използването на instance of ни казва , че имаме проблем с дизайна. В повечето случаи този проблем е, че не сме презаписали(override-нали) някой метод. И все пак съществуват крайни ситуации, в които използването на instanceOf е ок, но те са крайни.

КАТО ОБОБЩЕНИЕ МОЖЕ ДА СЕ КАЖЕ, ЧЕ ИЗПОЛЗВАНЕТО НА INSTANCE OF Е ЛОША ПРАКТИКА!!!

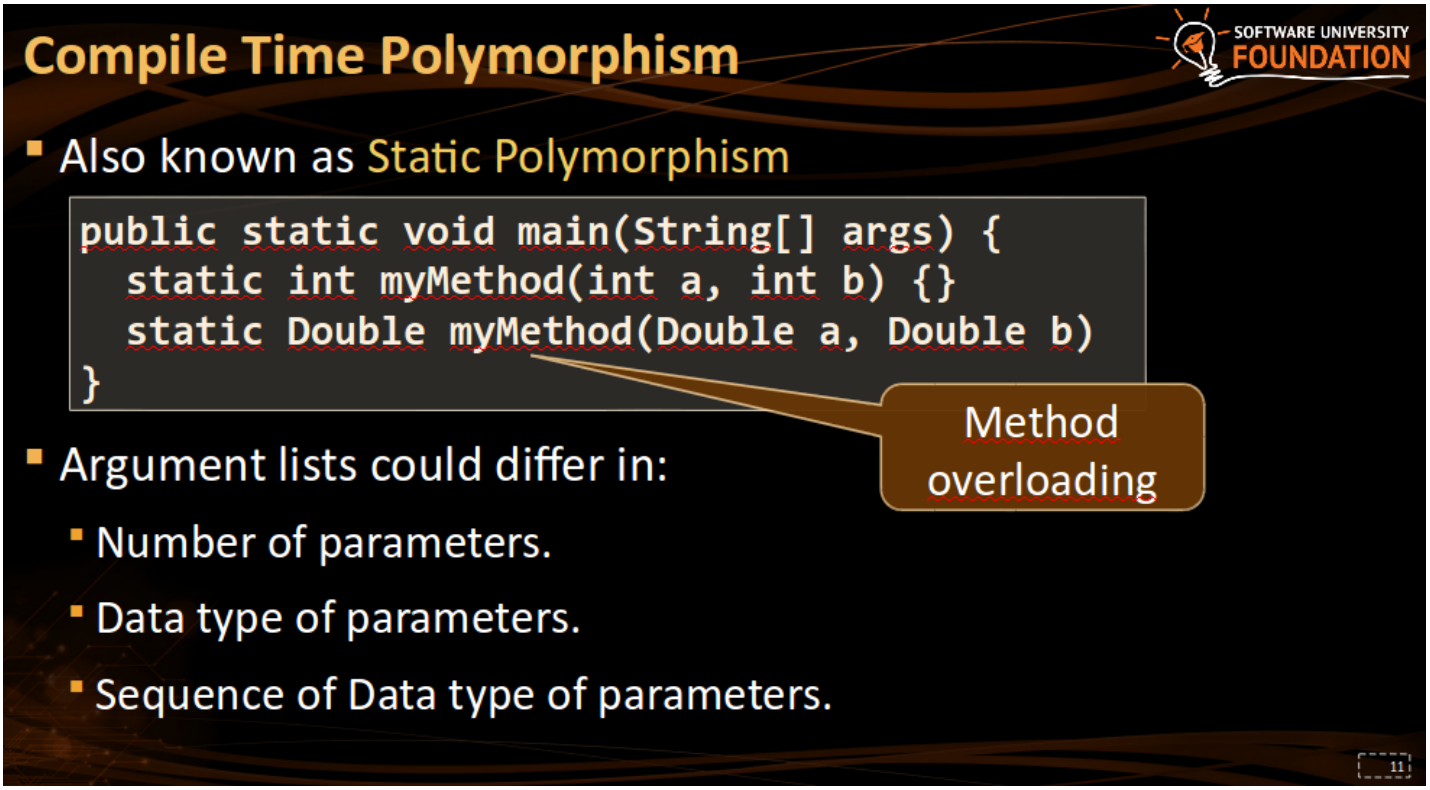
Types of polymorphism ? (Видове полиморфизъм)



Биват 2 вида:

- **Runtime polymorphism** - Полиморфизъм, при който компилаторът не може да разбере кога използваме полиморфизъм и в следствие на това проекта се пуска и някъде нещо гърми, защото нашият runtime polymorphism не е направен както трябва и throw-ва(хвърля) някакви exception-и (грешки), които не се знае откъде са и защо са там. Иначе казано , всеки път когато правим нещо от вида на **Shape shape = new Circle()**; както е показано на горната картинка, това е **Runtime polymorphism**, защото по време на компилация ако тук някъде има проблем, компилаторът няма да ни извести за такъв. **Runtime polymorphism-а залага главно на method overriding**. Тоест всяко дете(child) освен наследяването override-ва по-засуканите методи на своя родител.

- **Compile time polymorphism** - Представлява overload-ване на методи. Също така е познат и като Статичен полиморфизъм(**Static polymorphism**)

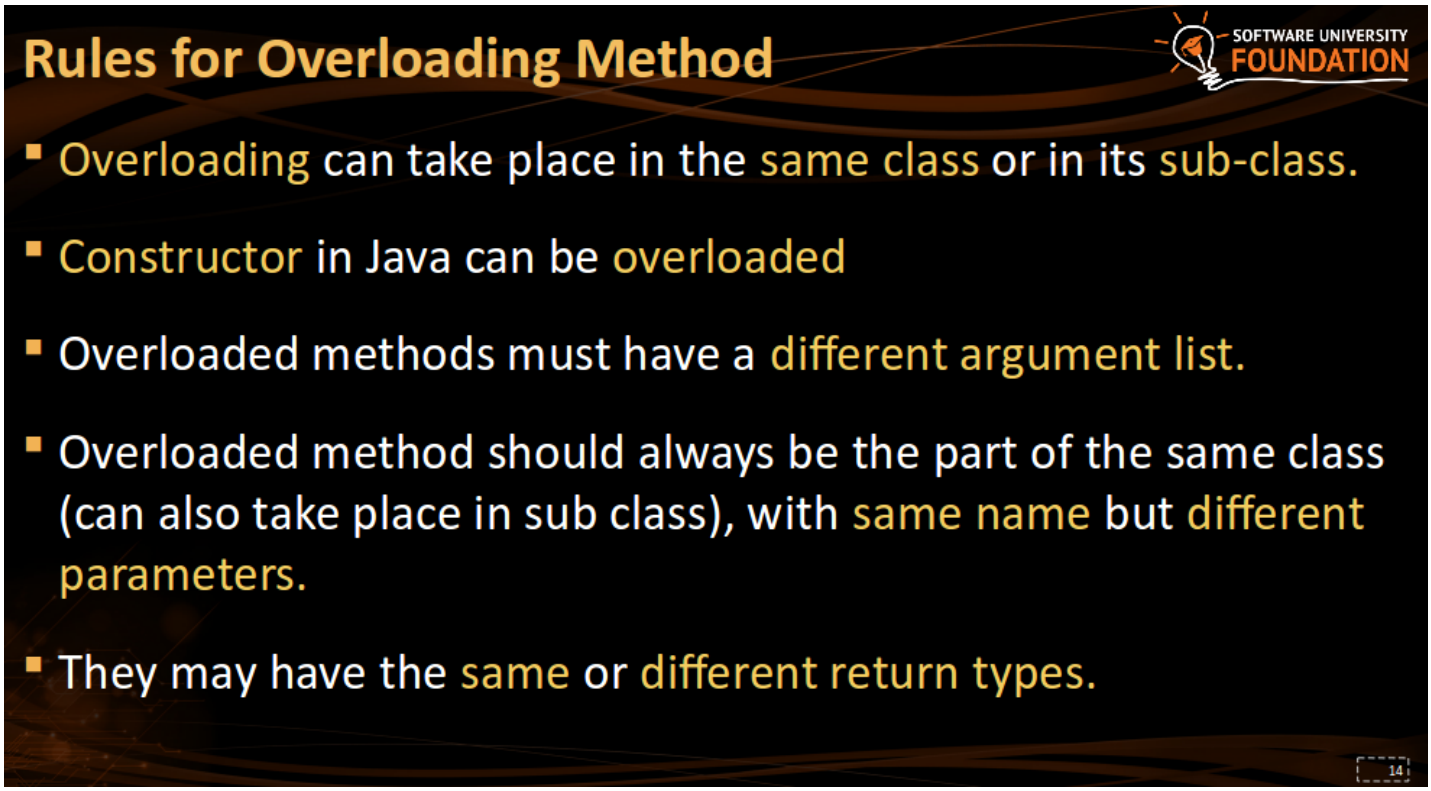


Това е, когато имаме метод с едно и също име, който искаме да се намери в един и същи клас.(Показано е на горната картинка). Това се постига като задължително трябва да имаме разлика в параметрите , които подаваме.**НЕ МОЖЕ ДА СА АБСОЛЮТНО ЕДНАКВИ!!!**.

Параметрите трябва да се различават по няколко неща:

> - ***тип***
> - ***брой***
> - ***в каква поредица.***

Rules for overloading method(Правила за overload-ване на методи)

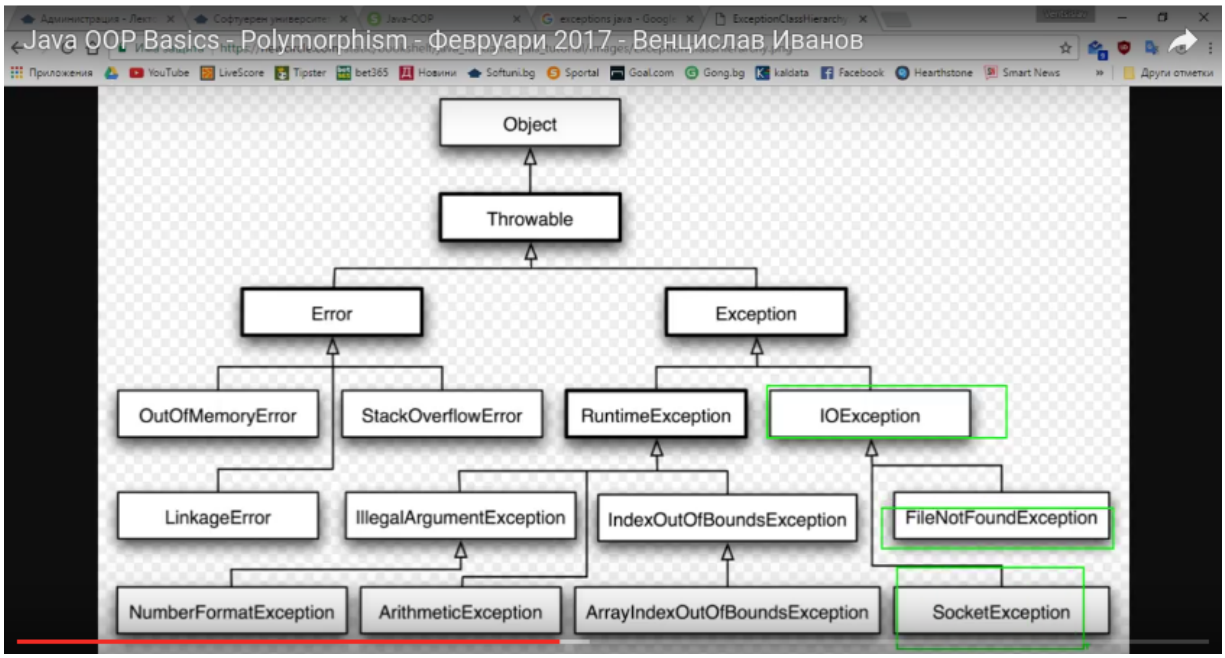


Какви са правилата за overload-ване на методи ?

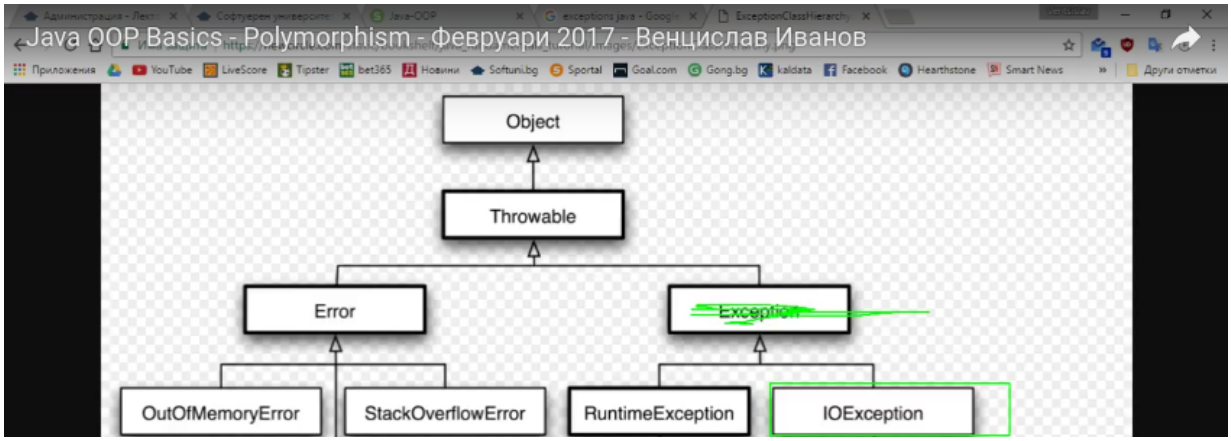
> - ***Overload-натия метод може да бъде в същия клас, но може да се направи*** *и в child класа.*
> *-Конструкторът също може да бъде overload-нат.*
> *-Overload-натите методи трябва да имат различен списък с аргументи.*
> *-Няма проблем да връщат един и същ тип данни. Return type-а може да бъде абсолютно еднакъв.*
> *-**НАЙ-ВАЖНОТО Е СПИСЪКА С АРГУМЕНТИ ДА БЪДЕ С РАЗЛИЧНИ ТИПОВЕ, КОЛИЧЕСТВО ИЛИ ПОСЛЕДОВАТЕЛНОСТ НА ПРОМЕНЛИВИТЕ, КОИТО ВКАРВАМЕ.***

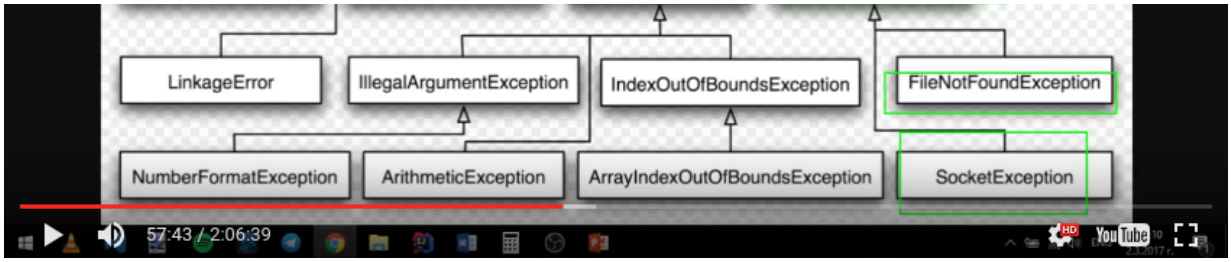
Какви са правилата за overrite-ване на методи ?

> - ***При overrite-ването методът трябва да е в child класа. Не може да е в*** *същия клас както при overload-ването. - За разлика от overload-ването, тук списъкът с аргументи трябва да е дословно един и същ. - Return type-ът трябва също да е еднакъв. Главният метод примерно връща int, overrite-натият метод също връща int. - Модификаторът за достъп може да бъде по-пропусклив, но не и по-притискащ. Това означава , че ако в родителския клас сме направили protected modifier-а, в детето не може да бъде private. Може да бъде само protected или public. - Private, static и final методите не могат да бъдат презаписвани. - Ако нашият метод хвърля exception в главният такъв, когато го overrite-ваме трябва задължително да хвърляме само exception-и, които вече сме прихванали(catch-нали) в нашия main метод. Примерно ако в главния клас хвърлим IOException примерно, това означава че когато наследим класа с overrite-натия метод в child класа, това означава , че можем да хвърляме долния и по-долния от IOExcecption exception, в случая - FileNotFoundException и Socket exception(Показано е на следващата картинка).*



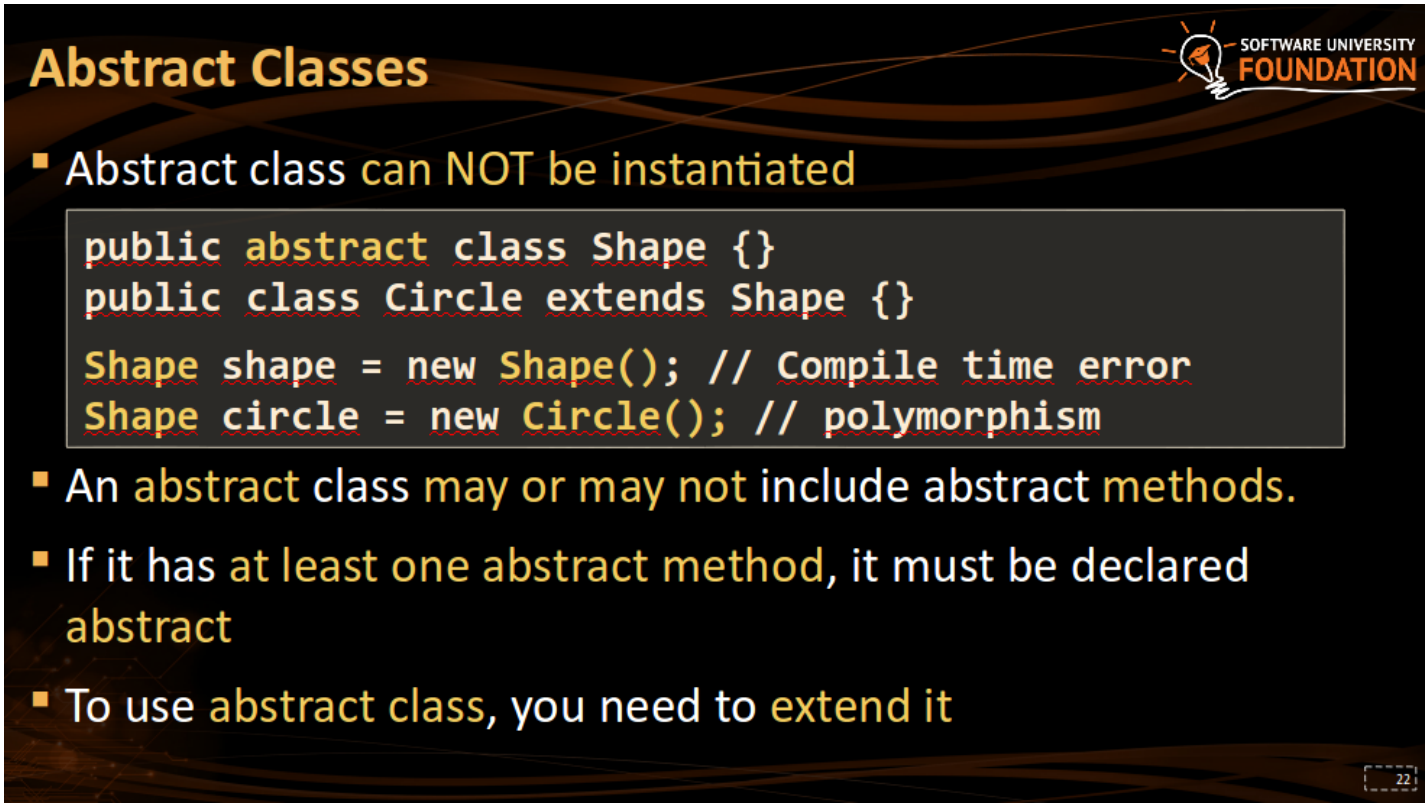
Но не можем да хвърлим родителския exception на IOException , който в случая се явява Exception. (Показано е на следващата картинка).**МОЖЕМ ДА ХВЪРЛЯМЕ САМО СЪЩИЯ ИЛИ ДЕТЕ НА НАШИЯ EXCEPTION**.



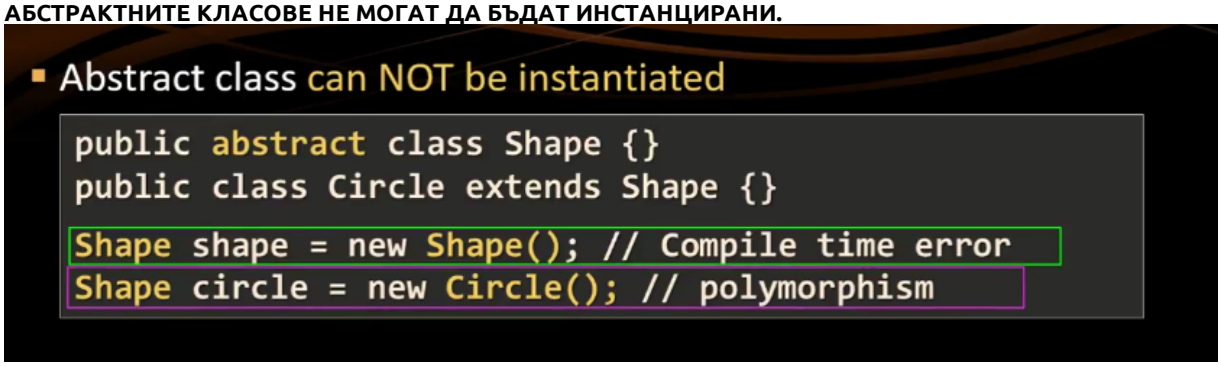


ПОНЯКОГА В КОДЪТ НИ СЕ НАЛАГА ДА ИЗПОЛЗВАМЕ НЕЩА, КОИТО НЕ СА КПК(КАЧЕСТВЕН ПРОГРАМЕН КОД) КАТО INSTANCE OF. ТРЯБВА ДА СЕ СТАРАЕМ ДА НЕ ГО ИЗПОЛЗВАМЕ. НЕ Е ОК ДА СЕ ИЗПОЛЗВА ОСВЕН В МНОГО, МНОГО КРАЕН СЛУЧАЙ. КАКВО ОБАЧЕ ДА НЕ ИЗПОЛЗВАМЕ В НИКАКЪВ СЛУЧАЙ ? АКО НЯКЪДЕ НИ СЕ НАЛОЖИ ЕДИН CHILD(КЛАС НАСЛЕДНИК) ПО НЯКАКВА ПРИЧИНА ДА ГО КАСТНЕМ КЪМ РОДИТЕЛСКИЯ КЛАС, ТОГАВА МОЖЕМ СПОКОЙНО ДА СИ НАБИЕМ НЯКОЛКО ШАМАРА. АКО НИ СЕ НАЛАГА ДА ПРАВИМ ТАКОВА НЕЩО, ТОВА ОЗНАЧАВА ЧЕ НАСЛЕДЯВАНЕТО НИ Е МНОГО ГРЕШНО. В НАШ КОД НЯМАМЕ НИКАКВА ОСНОВАТЕЛНА ПРИЧИНА ДА КАСТВАМЕ ДАДЕН ОБЕКТ КЪМ НЕГОВИЯ РОДИТЕЛ. АКО СЕ НАЛОЖИ ДА ИЗПОЛЗВАМЕ ТОВЯ НЯКЪДЕ, ОЗНАЧАВА ЧЕ СМЕ ОПЛЕСКАЛИ МНОГО РАБОТАТА. КАСТВАНЕ КЪМ РОДИТЕЛ Е ПО-КРАЕН СЛУЧАЙ ДОРИ И ОТ ИЗПОЛВАНЕТО НА INSTANCE OF. ТОВА ЧУПИ АБСОЛЮТНО ВСИЧКИ КОНВЕНЦИИ.

Abstract Classes

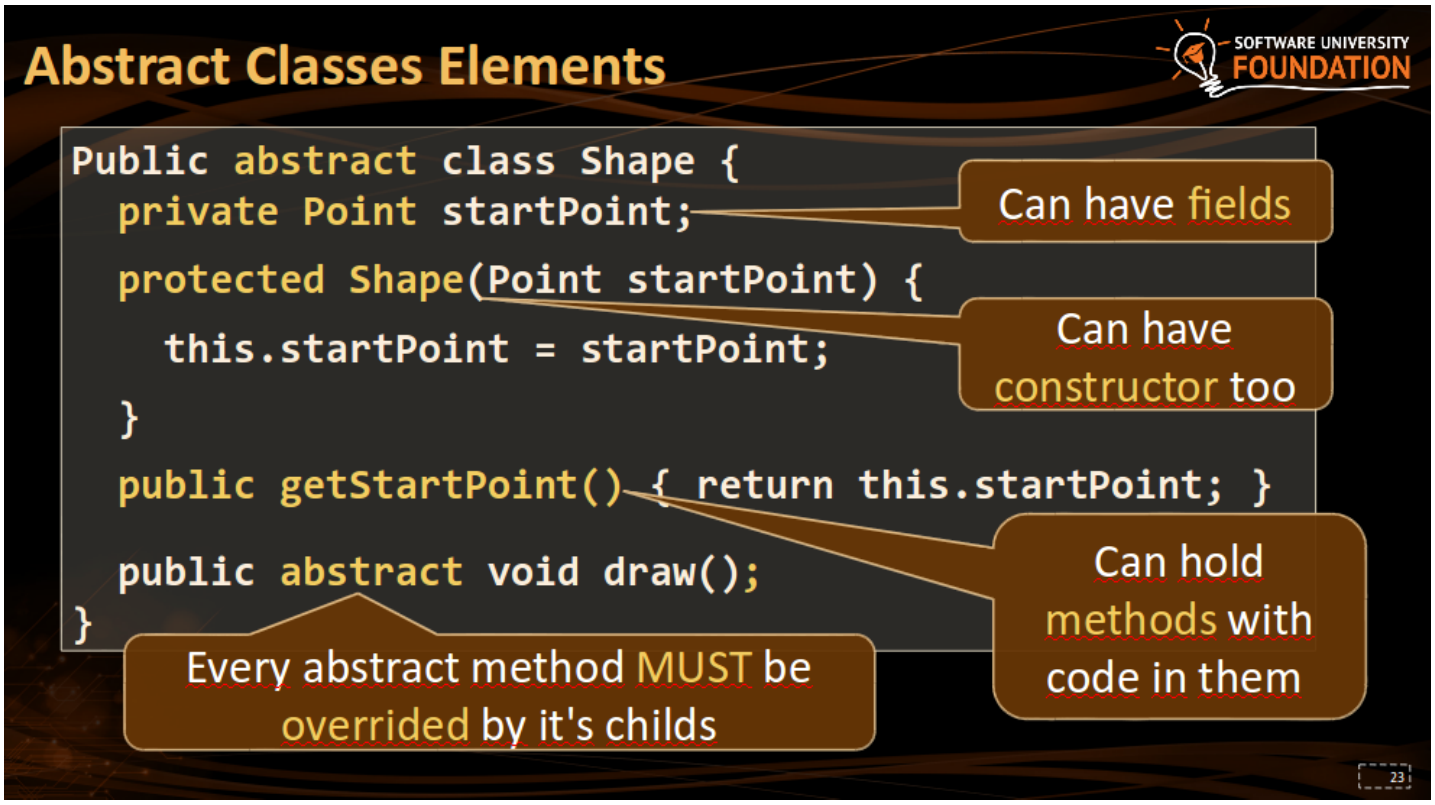


АБСТРАКТНИТЕ КЛАСОВЕ НЕ МОГАТ ДА БЪДАТ ИНСТАНЦИРАНИ.



Абстрактните класове мога да съдържат или не абстрактни методи.
Абстрактен метод е такъв без тяло, който когато бъде деклариран като абстрактен ,

**това в compile time веднага започва да реве ако класът не е абстрактен.
Когато правим абстрактен клас, за да можем да го използваме и да има някаква полза от него , по някакъв начин трябва да го extenend-нем(разширим, наследим). Иначе този клас само ще си стои и няма да можем да правим инстанции от него. Без extend-ване няма никакъв смисъл да създаваме абстрактен клас.**



Какво може да съдържа един абстрактен клас ?

> - ***Най-важното нещо е, че той може да съдържа поленца.***
> *- Въпреки че не се инстанцира няма никакъв проблем да си създадем конструктор, защото при наследниците можем да си извикаме супер конструктора\\\(родителския конструктор) и да записваме там данните, които да се записват в абстрактния клас.*
> *- Може да съдържа гетъри, сетъри или каквито там методи си искаме, които да имат логика в тях.*

Декларирането на абстрактен метод става чрез ключовата дума abstract. На картинката по-горе е показано как се дефинира правилно абстрактен клас. **ТАКЪВ МЕТОД ОБАЧЕ ЗАДЪЛЖАВА, ВСЕКИ НАСЛЕДНИК ДА ГО ПРЕЗАПИШЕ(OVERRIDE-НЕ).**