اگر با محتوای abstract اشنا باشید کلاسی مد نظرمون هست که نمیتونیم ازش نمونه ای رو بسازیم ولی میتونیم از اون ارثبری کنیم.

abstract class StoredEntity  
  
class Employee: StoredEntity**()**fun main**(**args: Array<String>**)** {  
 val se = Employee**()**}

خب حالا با خودتون میگین چه فایده ای داره که از کلاس های abstract استفاده کنیم؟ خب در واقع کلاس های abstract تفاوتی با کلاس های پایه ای دارند و اونم اینه که میتونیم state به مانند کلاس های معمولی داشته باشیم و همچنین میتونیم عضو های abstract داشته باشم. مثلا فرض کنید که چندین کلاس داریم که یک سری کار انجام میدن ولی محتوای هر کارشون اندکی متفاوته، برای همین میایم یک کلاس abstract میسازیم و تابع abstract اون رو هم میسازیم ولی بدنه تابع رو توی هر کدوم از کلاس ها بسته به نوع کلاسمون میسازیم.مثال زیر رو نگاه کنین:

abstract class StoredEntity{  
 abstract fun store**()**}  
  
class Employee: StoredEntity**()** {  
 override fun store**()** {  
 TODO**(**"not implemented"**)** }  
}

البته تموم عضو های کلاس abstract نیازی به abstract بودن خودشون ندارن:

abstract class StoredEntity{  
 val isActive = true  
 abstract fun store**()** fun status**()**:String{  
 return isActive.toString**()** }  
}  
  
class Employee: StoredEntity**()** {  
 override fun store**()** {  
 TODO**(**"not implemented"**)** }  
}  
  
fun main**(**args: Array<String>**)** {  
 val se = Employee**()** se.isActive  
 se.status**()**}

فقط همینطور که میدونید ما نمیتونیم توی main دوباره به isActive مقدار بدیم.