هدف این پروژه طراحی بازی othello است. این برنامه حاوی 4 کلاس است: کلاس Main که کلاس اصلی برنامه است، کلاس Disk که مربوط به هر مهره است، کلاس Map که مربوط به نقشه ی بازی است و کلاس player که کلاس بازیکن است.

کلاس Disk :

field های کلاس Disk شامل شماره ی ستون ، شماره ی سطر و یک کاراکتر تعیین کننده ی نوع آن دیسک است می باشد.
با دادن یک string از موقعیت و یک کاراکتر نوع دیسک میتوان یک object از این کلاس ساخت. در کانستراکتور این کلاس، رشته ی ورودی تبدیل به شماره ی ستون و شماره ی سطر میشوند.
متد های این کلاس شامل همه ی getter ها و نیز یک setter برای type است.

کلاس Map:

field این کلاس یک آرایه ی دو بعدی از کاراکتر است که نقشه ی اصلی بازی است. کانستراکتور این کلاس چیزی دریافت نمی کند و تنها یک نقشه ی 9 در 9 ایجاد کرده و با متد setMap (توضیحات آن در ادامه آمده) آن را مقدار دهی می کند.
متد های این کلاس شامل:

setMap: این متد سطر اول آرایه ی دو بعدی را با A,B,C,..,H و ستون اول آرایه ی دو بعدی را با 1,2,3,..,8 و سایر خانه ها را با کاراکتر + مقداردهی میکند.

updateMap: این متد map بازی را آپدیت میکند به گونه ای که ابتدا تمام خانه ها را از اول با صدا زدن متد setMap، از اول ست کرده و سپس سایر خانه ها را با آرایه ای که از تمام دیسک ها دریافت کرده مقداردهی می کند .

isFull: این متد بررسی می کند که آیا نقشه ی بازی پر است یا نه و در صورت پر بودن ، True برمیگرداند .
printMap:این متد تمامی خانه های آرایه ی دو بعدی را پرینت می کند .
printResult : این متد نتیجه ی بازی را چاپ می کند به گونه ای که تعداد خانه هایی که در آنها کاراکتر های سفید موجود است و نیز تعداد خانه هایی که در آنها کاراکتر سیاه موجود است را شمرده و هر کدام که بیشتر بود، آن را به عنوان برنده ی بازی معرفی می کند.
getChar : این متد کاراکتر موجود در شماره ی ستون و شماره ی سطر داده شده را برمی گرداند.

کلاس Player :

فیلد های این کلاس شامل یک کاراکتر برای نوع دیسک های یک بازیکن، یک آبجکت از کلاس Map برای نقشه ی بازی و یک ArrayList از کلاس Disk برای نگه داری دیسک های یک بازیکن.
کانستراکتور این کلاس یک کاراکتر برای تعیین نوع دیسک ها و یک Map برای نقشه ی بازی میگیرد.
متد های این کلاس شامل:
دو **getter** برای برگرداندن نوع دیسک ها و آرایه ی دیسک ها(getDisks و getDiskType)

addDisk: یک دیسک گرفته و آن را به لیست دیسک های کلاس اضافه میکند.

check_valid_position : یک دیسک جدید و یک لیست از دیسک های بازیکن دیگر گرفته و چک میکند که آیا مکان انتخاب شده برای قرار دادن آن دیسک جدید داده شده معتبر است یا خیر که در صورت معتبر بودن true و در غیر این صورت false برمیگرداند. عملکرد این متد به این گونه است که اگر در مکانی که دیسک جدید(new_disk) میخواهد قرار بگیرد ، یک دیسک از قبل حضور داشته باشد ، false برمیگرداند. سپس با پیمایش بر روی لیست دیسک های بازیکن دیگر چک میکند که آیا

دیسکی در این آرایه موجود است که با دیسک جدید(دیسک داده شده به عنوان ورودی) در یک ستون یا یک ردیف با فاصله ی 1 و یا در یک قطر با فاصله ی رادیکال 2 باشد یا نه. (در صورت وجود این دیسک را d1 می نامیم) اگر نبود false برمیگرداند.
سپس اگر جواب سوال قبل مثبت بود ، چک میکند که آیا در لیست دیسک های این بازیکن ، دیسکی وجود دارد که d1 بین آن دیسک و new_disk قرار بگیرد یا نه. در صورت وجود این دیسک را d2 می نامیم. اگر نبود false بر میگرداند.
سپس چک میکند که در فاصله ی بین new_disk و d2 خانه ی خالی یا مهره ی هم تایپ دیگری وجود دارد یا نه که اگر داشته باشد false بر میگرداند و اگر نداشته باشد یعنی همه ی شرایط برقرار است و با قرار دادن new_disk در جای خودش، حداقل یک دیسک از دیسک های بازیکن مقابل بین دو دیسک از این بازیکن قرار میگیرد.پس فقط در این صورت true بر میگرداند.

checkAndColor : یک دیسک جدید و یک لیست از دیسک های بازیکن مقابل گرفته و دیسک هایی که بین این دیسک جدید و یکی دیگر از دیسک های این بازیکن قرار گرفته اند را با صدا زدن متد color که توضیح آن در ادامه آمده است، به رنگ مشابه (تایپ مشابه) دیسک های این بازیکن در می آورد. (در اینجا برای واضح تر شدن از این بازیکن به عنوان بازیکن اول و از بازیکن مقابل به عنوان بازیکن دوم استفاده میشود.)
عملکرد این متد به گونه ای است که با انجام یک پیمایش بر روی لیست دیسک های بازیکن اول، به دنبال دیسکی میگردد که با دیسک جدید(در اینجا main_disk نامیده شده) در یک سطر یا ستون یا قطر باشد.
زمانی که این دیسک را یافت، بسته به نحوه ی قرار گرفتن این دو دیسک نسبت به هم، که همان طور که گفته شد می تواند در یک سطریا ستون یا قطر باشد ، مکان هایی که بین این دو دیسک قرار میگیرند را یافته و متد color را برای آن مکان صدا میزند که متد color بعد از پیدا کردن دیسکی از دیسک های بازیکن دوم که در آن مکان قرار گرفته ، آن دیسک را به رنگ سایر دیسک های بازیکن اول در می آورد.

color : این متد یک شماره ی سطر ، یک شماره ی ستون و یک لیست از دیسک های بازیکن دوم گرفته و با پیمایش بر روی لیست دیسک های بازیکن دوم ، دیسکی را میابد که در مکان داده شده قرار گرفته و رنگ آن را تغییر داده و از لیست دیسک های بازیکن دوم نیز آن را حذف می کند و به لیست دیسک های بازیکن اول اضافه میکند.

scanNewDisk : در این متد یک String اسکن شده و ابتدا چک میکند که این رشته معتبر هست یا نه به صورتی که اگر طول این رشته !=3 بود این رشته معتبرنیست و یا اگر بعد از اینکه به عدد تبدیل شد ، اعداد بدست آمده برای شماره ی سطر و ستون از 1 کوچکتر یا از 8 بزرگتر بودند، این position معتبر نیست.
در صورت معتبر بودن، این متد ابتدا با صدا زدن متد check_valid_position چک میکند که این موقعیت معتبر است یا نه.
اگر معتبر بود، این دیسک را به لیست دیسک های این بازیکن اضافه می کند و متد checkAndColor را صدا میزند.

availablePosition : این متد برای چک کردن این است که آیا خانه ای وجود دارد که بتوان در آن دیسکی برای بازیکن قرار داد یا نه. به عبارت دیگر آیا برای بازیکن حرکتی وجود دارد یا نه.عملکرد این متد به این گونه است که برای هر خانه یک دیسک جدید مد نظر گرفته شده و با فراخوانی متد
check_valid_positionبرای آن دیسک چک میشود که آیا آن دیسک میتواند درآن موقعیت قرار بگیرد یا نه و به محض پیدا کردن اولین خانه ای که بتوان دیسک جدیدی در آن قرار داد، true بر میگرداند و در غیر این صورت false برمیگرداند.

playerTurn :این متد یک لیست از دیسک های بازیکن مقابل را دریافت کرده و همه ی عملکرد هایی که باید در هر دور از بازی برای بازیکن انجام شود را پیاده سازی می کند.
عملکرد به گونه ای است که ابتدا با استفاده از متد availablePosition چک میکند که آیا برای بازیکن حرکتی ممکن است یا خیر. که در صورت نبودن حرکت عبارت pass را چاپ می کند.
اگر امکان انجام حرکتی برای بازیکن هست ، متد scanNewDisk را فراخوانی می کند و سپس یک لیست از همه ی دیسک های موجود درست کرده و map بازی راupdate می کند و سپس print می کند.

کلاس Main:

در این کلاس ابتدا یک آبجکت از کلاس map ایجاد می شود. سپس دو بازیکن ایجاد میشود که مهره های بازیکن اول مطابق بازی به رنگ سیاه و مهره های بازیکن دوم به رنگ سفید است.(البته اگر پشت زمینه ی terminal یا console سفید باشد جای این دو با هم عوض می شود!!)
سپس 4 دیسک اولیه ی بازی ایجاد شده و نسخه ی اولیه ی map، آپدیت و print می شود. بعد از آن وارد یک حلقه ی true می شویم که در ابتدای آن چک می شود که آیا نقشه ی بازی پر شده است یا نه(با فراخوانی map.isFull) که در صورت پر شدن حلقه شکسته می شود.
سپس چک میشود که آیا برای بازیکن ها حرکتی موجود است یا نه (با فراخوانی availablePosition برای هر بازیکن) که در صورت نبود حرکت برای هر دو بازیکن ، حلقه شکسته می شود.
سپس تابع playerTurn ابتدا برای بازیکن اول و سپس برای بازیکن دوم صدا زده می شود و این چرخه ادامه پیدا می کند تا جایی که شکسته شود.
بعد از شکسته شدن حلقه که همان پایان بازی است، نقشه ی بازی یکبار دیگرپرینت شده و نتیجه ی بازی نیز چاپ میشود.