کلاس های اصلی:

- **Card:** این کلاس abstract، کلاس پایه برای انواع کارت های بازی (اتاق، شخص، و وسیله) است.
- **Room، Place، Person:** این کلاس ها از کلاس Card ارث بری می کنند و هر کدام نشان دهنده یک نوع کارت در بازی هستند.
- **Davar:** این کلاس، اطلاعات مربوط به پاسخ معما (دزد، اتاق ، و مکان الماس) را در خود نگه می دارد.
- **Player:** این کلاس، بازیکنان بازی را مدل سازی می کند و شامل کارت های هر بازیکن، اطلاعات مربوط به مکان فعلی آن ها در بازی، و متدهایی برای پرسیدن سوال و پاسخ دادن به سوالات است.
- **Game:** این کلاس، مدیریت کلی بازی را بر عهده دارد. شامل لیست بازیکنان، کارت ها، و متدهایی برای شروع بازی، توزیع کارت ها، و انجام مراحل بازی است.

متدهای مهم:

- **ask:** این متد در کلاس Player، برای پرسیدن سوال از سایر بازیکنان استفاده می شود. بازیکن با توجه به تاس هایی که انداخته است، یک اتاق، یک شخص، و یک وسیله را انتخاب می کند و از سایر بازیکنان می پرسد که آیا کارت های مربوطه را دارند یا خیر.
- **answer:** این متد در کلاس Player، برای پاسخ دادن به سوالات سایر بازیکنان استفاده می شود. اگر بازیکن کارت های مورد نظر را داشته باشد، یکی از آن ها را به بازیکن سوال کننده نشان می دهد.
- **is_correct:** این متد در کلاس Davar، بررسی می کند که آیا حدس بازیکن در مورد پاسخ معما درست است یا خیر.
- **gues:** این متد در کلاس Game، یک نوبت از بازی را انجام می دهد. در هر نوبت، یک بازیکن تاس می اندازد، سوال می پرسد، و سایر بازیکنان به سوال او پاسخ می دهند.
- **main:** این متد، نقطه شروع اجرای بازی است. یک نمونه از کلاس Game ایجاد می کند و بازی را شروع می کند.

توضیح مختصر درباره منطق بازی:

1. **شروع بازی:** کارت ها بین بازیکنان توزیع می شوند و کارت های پاسخ معما به صورت تصادفی انتخاب و از سایر کارت ها جدا می شوند.

2. **نوبت بازیکن:** هر بازیکن در نوبت خود تاس می اندازد و با توجه به نتیجه تاس، می تواند به اتاق های مختلف برود. سپس می تواند در مورد پاسخ معما حدس بزند یا از سایر بازیکنان سوال بپرسد.
3. **پرسیدن سوال:** بازیکن یک اتاق، یک شخص، و یک مکان را انتخاب می کند و از سایر بازیکنان می پرسد که آیا کارت های مربوطه را دارند یا خیر.
4. **پاسخ دادن به سوال:** اگر بازیکنی کارت های مورد نظر را داشته باشد، یکی از آن ها را به بازیکن سوال کننده نشان می دهد.
5. **حدس زدن پاسخ معما:** بازیکن می تواند در هر نوبت، در مورد پاسخ معما حدس بزند. اگر حدس او درست باشد، برنده بازی است.
6. **پایان بازی:** بازی تا زمانی ادامه پیدا می کند که یک بازیکن حدس درست بزند یا تمام بازیکنان به جز یک نفر از دور بازی خارج شوند.

کلاس Card

یک کلاس انتزاعی (abstract) در جاوا است که به عنوان یک قالب یا الگوی کلی برای انواع کارت‌های بازی در نظر گرفته شده است. این کلاس به تنهایی قابل استفاده نیست و باید از آن کلاس‌های دیگری ایجاد شوند (ارث‌بری) که هر کدام نشان‌دهنده یک نوع خاص از کارت هستند (مانند کارت اتاق، کارت شخصیت، یا کارت مکان).

ویژگی‌ها:

- **انتزاعی بودن (abstract):** این کلاس نمی‌تواند به‌طور مستقیم نمونه‌سازی شود (یعنی نمی‌توانید یک شیء از نوع Card ایجاد کنید). این کلاس فقط یک طرح کلی برای کارت‌ها ارائه می‌دهد و جزئیات مربوط به هر نوع کارت در کلاس‌های فرزند آن مشخص می‌شود.
- **کپسوله‌سازی (Encapsulation):** متغیر name که نام کارت را در خود ذخیره می‌کند، به صورت خصوصی (private) تعریف شده است. این به این معنی است که این متغیر فقط از داخل خود کلاس Card قابل دسترسی و تغییر است. برای دسترسی به نام کارت از بیرون از کلاس، باید از متد getName() استفاده کرد.

متدها:

- **سازنده (Constructor):** این متد (Card(String name)) هنگام ایجاد یک شیء از کلاس‌های فرزند Card فراخوانی می‌شود. این متد نام کارت را دریافت کرده و آن را در متغیر name ذخیره می‌کند.
- **getName():** این متد یک رشته (String) را برمی‌گرداند که نام کارت را نشان می‌دهد.
- **getCardType():** این یک متد انتزاعی است. متدهای انتزاعی بدنه ندارند و باید در کلاس‌های فرزند پیاده‌سازی شوند. هدف این متد این است که در هر کلاس فرزند، نوع کارت (مثلاً "اتاق"، "شخص" یا "مکان") را مشخص کند.

کلاس‌های فرزند:

کلاس‌های دیگری مانند Room (اتاق)، Person (شخص) و Place (مکان) از کلاس Card ارث‌بری می‌کنند. این کلاس‌های فرزند، متد getCardType() را پیاده‌سازی می‌کنند تا نوع کارت خود را مشخص کنند. به عنوان مثال، کلاس Room این متد را طوری پیاده‌سازی می‌کند که رشته "Room" را برگرداند.

مزایای استفاده از کلاس انتزاعی و ارث‌بری:

- **قابلیت استفاده مجدد کد:** کدهای مشترک بین انواع کارت‌ها (مانند متغیر name و متد getName()) در کلاس Card نوشته می‌شوند و در کلاس‌های فرزند قابل استفاده مجدد هستند.

- **پیمانه‌ای بودن (Modularity):** کد به بخش‌های کوچک‌تر و قابل مدیریت‌تری تقسیم می‌شود که باعث افزایش خوانایی و نگهداری آسان‌تر کد می‌شود.
- **انعطاف‌پذیری:** با اضافه کردن کلاس‌های فرزند جدید، می‌توان انواع کارت‌های جدید را به بازی اضافه کرد، بدون اینکه نیاز به تغییر کدهای قبلی باشد.

کلاس Room

نمایانگر مفهوم یک اتاق در بازی "سرنخ" (Cluedo) است. این کلاس از کلاس پایه Card ارث‌بری می‌کند، به این معنی که یک اتاق به عنوان یک نوع کارت در نظر گرفته می‌شود.

ویژگی‌ها:

- **n (شماره اتاق):** یک عدد صحیح (int) است که به طور منحصر به فرد هر اتاق را در بازی مشخص می‌کند. این شماره می‌تواند برای ردیابی مکان بازیکن یا تعیین اتاق‌های مجاز برای حرکت استفاده شود.
- **name (نام اتاق):** یک رشته (String) است که نام اتاق (مثلاً "کتابخانه"، "آشپزخانه" یا "اتاق نشیمن") را در خود ذخیره می‌کند. این نام برای نمایش به بازیکن و اهداف دیگر در بازی استفاده می‌شود.

سازنده (Constructor):

- Room(String name, int n): این سازنده دو پارامتر ورودی می‌گیرد: نام اتاق (name) و شماره اتاق (n).
  - ابتدا، سازنده کلاس پدر ((super(name) را فراخوانی می‌کند تا نام کارت را مقداردهی اولیه کند. این کار با استفاده از کلمه کلیدی super انجام می‌شود که به سازنده کلاس پدر اشاره دارد.
  - سپس، مقدار پارامتر n را در متغیر n شیء Room ذخیره می‌کند.

متدها:

- getN(): این متد به سادگی مقدار متغیر n (شماره اتاق) را برمی‌گرداند.
- getCardType(): این متد بازنویسی شده از کلاس پدر (Card) است. این متد نوع کارت را مشخص می‌کند و در این کلاس همواره مقدار "Room" را برمی‌گرداند. این کار با استفاده از حاشیه‌نویسی @Override مشخص شده است که به کامپایلر جاوا می‌گوید این متد قصد بازنویسی یک متد از کلاس پدر را دارد.

ارتباط با کلاس Card:

کلاس Room با ارث‌بری از کلاس Card، به طور خودکار ویژگی name (نام کارت) و متد getName() (برای دریافت نام کارت) را از کلاس Card به ارث می‌برد. این کار باعث می‌شود تا کد تمیزتر و قابل فهم‌تر شود و از تکرار کد جلوگیری شود.

استفاده در بازی:

اشیاء کلاس Room برای نمایش اتاق‌های مختلف در بازی "سرنخ" استفاده می‌شوند. هر اتاق دارای یک نام و یک شماره منحصر به فرد است. بازیکنان در طول بازی می‌توانند بین اتاق‌ها حرکت کنند و با توجه به کارت‌هایی که در دست دارند، حدس بزنند که دزدی در کدام اتاق اتفاق افتاده است.

کلاس Person

نمایانگر مفهوم یک شخص یا مظنون در بازی "سرنخ" (Cluedo) است. این کلاس از کلاس پایه Card ارث‌بری می‌کند، به این معنی که یک شخص به عنوان یک نوع کارت در نظر گرفته می‌شود.

ویژگی‌ها:

- **name (نام شخص):** یک رشته (String) را در خود ذخیره می‌کند. این نام برای نمایش به بازیکن و اهداف دیگر در بازی استفاده می‌شود.

سازنده (Constructor):

- Person(String name): این سازنده یک پارامتر ورودی می‌گیرد: نام شخص (name).
  - ابتدا، سازنده کلاس پدر ((super(name)) را فراخوانی می‌کند تا نام کارت را مقداردهی اولیه کند.

متدها:

- getCardType(): این متد بازنویسی شده از کلاس پدر (Card) است. این متد نوع کارت را مشخص می‌کند و در این کلاس همواره مقدار "Person" را برمی‌گرداند.

ارتباط با کلاس Card:

کلاس Person با ارث‌بری از کلاس Card، به طور خودکار ویژگی name (نام کارت) و متد getName() (برای دریافت نام کارت) را از کلاس Card به ارث می‌برد.

استفاده در بازی:

اشیاء کلاس Person برای نمایش مظنونین مختلف در بازی "سرنخ" استفاده می‌شوند. هر مظنون دارای یک نام منحصر به فرد است. بازیکنان در طول بازی می‌توانند با توجه به کارت‌هایی که در دست دارند، حدس بزنند که کدام مظنون دزد است.

کلاس Place

نمایانگر مفهوم یک مکان در بازی "سرنخ" (Cluedo) است. این کلاس از کلاس پایه Card ارث‌بری می‌کند، به این معنی که یک مکان به عنوان یک نوع کارت در نظر گرفته می‌شود.

ویژگی‌ها:

- **name (نام مکان):** یک رشته (String) است که نام مکان (مثلاً زیر گلدان، آشپزخانه یا اتاق نشیمن) را در خود ذخیره می‌کند. این نام برای نمایش به بازیکن و اهداف دیگر در بازی استفاده می‌شود.

سازنده (Constructor):

- Place(String name): این سازنده یک پارامتر ورودی می‌گیرد: نام مکان (name).
  - ابتدا، سازنده کلاس پدر ((super(name)) را فراخوانی می‌کند تا نام کارت را مقداردهی اولیه کند.

متدها:

- getCardType(): این متد بازنویسی شده از کلاس پدر (Card) است. این متد نوع کارت را مشخص می‌کند و در این کلاس همواره مقدار "Place" را برمی‌گرداند.

ارتباط با کلاس Card:

کلاس Place با ارث‌بری از کلاس Card، به طور خودکار ویژگی name (نام کارت) و متد getName() (برای دریافت نام کارت) را از کلاس Card به ارث می‌برد.

استفاده در بازی:

اشیاء کلاس Place برای نمایش مکان‌های مختلف در بازی "سرنخ" استفاده می‌شوند. هر مکان دارای یک نام منحصر به فرد است. بازیکنان در طول بازی می‌توانند با توجه به کارت‌هایی که در دست دارند، حدس بزنند که دزدی در کدام مکان اتفاق افتاده است.

کلاس Davar

در بازی "سرنخ" به عنوان نگه دارنده پاسخ نهایی بازی عمل می کند. این کلاس اطلاعات مربوط به اتاقی که جرم در آن رخ داده، شخص دزد و محل اختفای الماس را در خود نگه می دارد. به عبارت دیگر، این کلاس حکم داور یا قاضی را در بازی دارد و درستی یا نادرستی حدس های بازیکنان را تعیین می کند.

ویژگی ها:

- **name:** یک رشته (String) برای نگهداری نام. این ویژگی در نسخه فعلی کد استفاده نمی شود، اما می تواند برای توسعه های آینده بازی مفید باشد .
- **room:** یک شیء از کلاس Room که اتاق محل وقوع جرم را نشان می دهد.
- **person:** یک شیء از کلاس Person که شخص دزد را نشان می دهد.
- **place:** یک شیء از کلاس Place که محل اختفای الماس را نشان می دهد.

متدها:

- **Davar(String name, Room room, Person person, Place place) (سازنده):** این متد برای مقداردهی اولیه ویژگی های کلاس استفاده می شود. هنگام ایجاد یک شیء از کلاس Davar، نام ، اتاق، شخص و مکان به عنوان ورودی دریافت می شوند و در ویژگی های مربوطه ذخیره می شوند.
- **is_correct(Room room, Person person, Place place):** این متد برای بررسی درستی حدس بازیکن استفاده می شود. این متد سه پارامتر ورودی می گیرد که نشان دهنده حدس بازیکن در مورد اتاق، شخص و مکان هستند. سپس نام این موارد را با مقادیر ذخیره شده در ویژگی های کلاس مقایسه می کند. اگر هر سه مقدار با هم تطابق داشته باشند، عدد 1 را برمی گرداند (به معنی حدس درست)، در غیر این صورت 0 را برمی گرداند (به معنی حدس نادرست).
- **getName():** این متد نام دزد را برمی گرداند.

- **toString():** این متد اطلاعات مربوط به پاسخ نهایی بازی (نام اتاق، شخص و مکان) را به صورت یک رشته فرمت شده برمی گرداند. این رشته می تواند برای نمایش پاسخ نهایی در پایان بازی استفاده شود.

نحوه استفاده در بازی:

در ابتدای بازی، یک شیء از کلاس Davar ایجاد می شود و اطلاعات مربوط به پاسخ نهایی بازی به آن داده می شود. سپس در طول بازی، هر زمان که یک بازیکن حدسی می زند، متد is_correct فراخوانی می شود تا درستی حدس او بررسی شود. در پایان بازی، اطلاعات پاسخ نهایی با استفاده از متد toString به بازیکنان نمایش داده می شود.

کلاس Player

در بازی "سرنخ" (Cluedo) نماینده هر بازیکن در بازی است و اطلاعات مربوط به بازیکن، کارت‌های او و اقدامات او را مدیریت می‌کند.

ویژگی‌ها (Attributes):

1. **name (نام بازیکن):** یک رشته (String) است که نام بازیکن را در خود ذخیره می‌کند.
2. **cards (لیست کارت‌های بازیکن):** یک لیست (List) از اشیاء کلاس Card است که کارت‌های موجود در دست بازیکن را نگهداری می‌کند.
3. **cards_number (تعداد کارت‌های بازیکن):** یک عدد صحیح (int) است که تعداد کارت‌های موجود در دست بازیکن را نشان می‌دهد.
4. **room_number (شماره اتاق فعلی بازیکن):** یک عدد صحیح (int) است که نشان می‌دهد بازیکن در حال حاضر در کدام اتاق قرار دارد.
5. **places (لیست کارت‌های مکان بازیکن):** یک لیست (List) از اشیاء کلاس Place است که کارت‌های مکان‌هایی که بازیکن دیده است را نگهداری می‌کند.
6. **persons (لیست کارت‌های شخص بازیکن):** یک لیست (List) از اشیاء کلاس Person است که کارت‌های اشخاصی که بازیکن دیده است را نگهداری می‌کند.
7. **rooms (لیست کارت‌های اتاق بازیکن):** یک لیست (List) از اشیاء کلاس Room است که کارت‌های اتاق‌هایی که بازیکن دیده است را نگهداری می‌کند.

سازنده‌ها (Constructors):

1. **Player(String name, List<Card> cards)**

این سازنده یک بازیکن جدید با نام و لیست کارت‌های اولیه ایجاد می‌کند. کارت‌ها را در لیست cards ذخیره کرده و تعداد آن‌ها را در cards_number قرار می‌دهد. همچنین، اتاق فعلی بازیکن را به اتاق شروع (شماره 1) تنظیم می‌کند و لیست‌های places، persons و rooms را برای ردیابی کارت‌های دیده شده ایجاد می‌کند. سپس، کارت‌های اولیه بازیکن را بررسی کرده و آن‌ها را بر اساس نوعشان (مکان، شخص یا اتاق) در لیست‌های مربوطه قرار می‌دهد.

2. **Player(String name):** این سازنده یک بازیکن جدید فقط با نام ایجاد می‌کند. در این حالت، لیست کارت‌های بازیکن خالی است و تعداد کارت‌ها صفر است. لیست‌های places، persons و rooms نیز خالی ایجاد می‌شوند.

متدها (Methods):

1. **addCard(Card card):** این متد یک کارت جدید را به لیست کارت‌های بازیکن (cards) اضافه می‌کند، تعداد کارت‌ها (cards_number) را یک واحد افزایش می‌دهد و سپس کارت را با استفاده از متد addCard_to_data به لیست مربوط به نوع آن اضافه می‌کند.
2. **addCard_to_data(Card card):** این متد کارت ورودی را بررسی می‌کند و آن را به لیست مربوط به نوع آن (مکان، شخص یا اتاق) اضافه می‌کند.
3. **getCards():** این متد لیست کارت‌های بازیکن (cards) را برمی‌گرداند.
4. **getCards_number():** این متد تعداد کارت‌های بازیکن (cards_number) را برمی‌گرداند.
5. **getRoomNumber():** این متد شماره اتاق فعلی بازیکن (room_number) را برمی‌گرداند.
6. **have_this_card(Card card):** این متد بررسی می‌کند که آیا بازیکن کارت ورودی را در دست دارد یا خیر. اگر کارت در لیست کارت‌های بازیکن وجود داشته باشد، 1 را برمی‌گرداند و در غیر این صورت 0 را برمی‌گرداند.
7. **is_valid_room(int newRoom, int tas1, int tas2):** این متد بررسی می‌کند که آیا رفتن به اتاق جدید (newRoom) با توجه به مقادیر تاس‌ها (tas1 و tas2) و قوانین بازی معتبر است یا خیر. اگر حرکت معتبر باشد، 1 را برمی‌گرداند و در غیر این صورت 0 را برمی‌گرداند.
8. **ask(int tas1, int tas2, List<Room> allrooms_, List<Person> allpersons_, List<Place> allplaces_, List<Card> newCards)**

این متد برای انتخاب کارت‌های اتاق، شخص و مکان مورد نظر برای پرسیدن سوال استفاده می‌شود (نسخه خودکار).

9. **ask(int tas1, int tas2, List<Room> allrooms, List<Person> allpersons, List<Place> allplaces, List<Card> newCards, Scanner scanner)**

این متد برای انتخاب کارت‌های اتاق، شخص و مکان مورد نظر برای پرسیدن سوال استفاده می‌شود (نسخه تعاملی با ورودی کاربر).

10. **final_gues(List<Room> allrooms, List<Person>allpersons, List<Place> allplaces, List<Card> newCards, Davar davar)**

این متد برای انجام حدس نهایی در بازی استفاده می‌شود (نسخه خودکار).

11. **allpersons, <Person>allrooms, List <Room>final_gues(List <Place>List allplaces, Davar davar, Scanner scanner):** این متد برای انجام حدس نهایی در بازی استفاده می‌شود (نسخه تعاملی با ورودی کاربر).

12. **answer(List<Card> newCards, Player other)**

این متد برای پاسخ دادن به سوال بازیکن دیگر استفاده می‌شود. اگر بازیکن کارت مورد نظر را داشته باشد، آن را به بازیکن دیگر نشان می‌دهد و در غیر این صورت اعلام می‌کند که کارت را ندارد.

13. **getName():** این متد نام بازیکن را برمی‌گرداند.

14. **show_data():** این متد اطلاعات مربوط به کارت‌های بازیکن (اتاق‌ها، اشخاص و مکان‌ها) را به صورت یک رشته فرمت شده برمی‌گرداند.

15. **toString():** این متد اطلاعات کامل بازیکن (نام، تعداد کارت‌ها، لیست کارت‌ها، اتاق‌ها، اشخاص و مکان‌ها) را به صورت یک رشته فرمت شده برمی‌گرداند.

کلاس Game

نقش اصلی در مدیریت و اجرای بازی "سرنخ" را بر عهده دارد. این کلاس شامل اطلاعات مربوط به بازیکنان، کارت‌ها (اتاق، شخص، مکان)، تاس‌ها، داور بازی و وضعیت بازی (شروع، پایان، برنده) است.

ویژگی‌ها (Attributes):

1. **persons:** لیستی از اشیاء کلاس Person که نمایانگر شخصیت‌های بازی هستند.
2. **places:** لیستی از اشیاء کلاس Place که نمایانگر مکان‌های مختلف در بازی هستند.
3. **rooms:** لیستی از اشیاء کلاس Room که نمایانگر اتاق‌های بازی هستند.
4. **players:** لیستی از اشیاء کلاس Player که نمایانگر بازیکنان بازی هستند.
5. **tas:** لیستی از اعداد صحیح که مقادیر ممکن تاس‌ها را نشان می‌دهد (1 تا 6).
6. **davar:** یک شیء از کلاس Davar که نقش داور بازی را دارد و پاسخ نهایی معما را می‌داند.
7. **end_game:** یک متغیر صحیح که نشان می‌دهد آیا بازی تمام شده است یا خیر (0: بازی ادامه دارد، 1: بازی تمام شده است).
8. **play:** یک متغیر بولی که نشان می‌دهد آیا بازیکن انسانی در حال بازی است یا خیر (true: بله، false: خیر).
9. **palayer_number:** یک متغیر صحیح که تعداد بازیکنان در بازی را نشان می‌دهد.

سازنده (Constructor):

- Game(int palayer_number, boolean play): این سازنده، یک بازی جدید را با تعداد بازیکنان مشخص (palayer_number) و حالت بازی (play) ایجاد می‌کند. در این متد، ابتدا لیست‌های مربوط به اشخاص، مکان‌ها، اتاق‌ها، بازیکنان و تاس‌ها مقداردهی اولیه می‌شوند. سپس با فراخوانی متد add_data، اطلاعات اولیه بازی (اشخاص، مکان‌ها و اتاق‌ها) به لیست‌ها اضافه می‌شود. همچنین، داور بازی (davar) ایجاد شده و کارت‌های پاسخ نهایی به صورت تصادفی انتخاب می‌شوند. در نهایت، کارت‌ها بین بازیکنان توزیع می‌شوند و ترتیب بازیکنان به صورت تصادفی تعیین می‌شود.

متدها (Methods):

1. **gues(int n, Scanner scanner):** این متد، نوبت یک بازیکن را مدیریت می‌کند. ابتدا بازیکن فعلی (p) بر اساس شماره نوبت (n) و تعداد بازیکنان (palayer_number) مشخص می‌شود. سپس دو تاس به صورت تصادفی انداخته می‌شوند و یک لیست خالی برای کارت‌های جدید ایجاد می‌شود. اگر بازیکن انسانی است (play = true) و نوبت بازیکن اول است، از او خواسته می‌شود که بین پرسیدن از داور یا یک بازیکن دیگر یکی را انتخاب کند. در غیر این صورت، بازیکن به صورت خودکار حدس نهایی را می‌زند یا سوالی می‌پرسد. در نهایت، پاسخ سایر بازیکنان بررسی می‌شود و در صورت لزوم، اطلاعات بازیکن فعلی چاپ می‌شود.

2. **add_data():** این متد، اطلاعات اولیه بازی (اشخاص، مکان‌ها و اتاق‌ها) را به لیست‌های مربوطه اضافه می‌کند.
3. **addPerson(Person person):** این متد، یک شخص جدید را به لیست اشخاص (persons) اضافه می‌کند.
4. **addPlace(Place place):** این متد، یک مکان جدید را به لیست مکان‌ها (places) اضافه می‌کند.
5. **addRoom(Room room):** این متد، یک اتاق جدید را به لیست اتاق‌ها (rooms) اضافه می‌کند.
6. **addPlayer(Player player):** این متد، یک بازیکن جدید را به لیست بازیکنان (players) اضافه می‌کند.
7. **getPlayers():** این متد، لیست بازیکنان (players) را برمی‌گرداند.

متد main:

- در متد main، یک شیء از کلاس Game با 3 بازیکن و حالت بازی true ایجاد می‌شود. سپس یک شیء Scanner برای دریافت ورودی از کاربر ایجاد می‌شود. یک حلقه while برای اجرای بازی تا زمانی که بازی تمام نشده باشد (end_game == 0) اجرا می‌شود. در هر تکرار حلقه، متد gues برای بازیکن فعلی فراخوانی می‌شود و شمارنده نوبت (n) افزایش می‌یابد. در نهایت، شیء Scanner بسته می‌شود.

چالش ها و راه حل ها

1. انتخاب رندوم و تصادفی

برای اینکه از پیچیدگی های انتخاب اعداد تصادفی و تکراری نبودن آنها جلو گیری کنیم از شافل کردن استفاده میکنیم.

2. حدث زدن بازیکنان

ایجاد برگه دیتا برای هر بازیکن و پرکردن آن به صورتی که هر بازیکن یک کارت را دید آن کارت را به دیتا های خودش اضافه میکند و زمانی که فقط 1 کارت از هر نوع کارت ماند حدث نهایی را میزند.

3. برسی اتفاقات داخل کد

برای اینکه متوجه بشیم کد درست کار میکند تمامی اطلاعات هر بازین و کارت های مخفی را چاپ میکنیم تا عملکرد تمام کد را ببینیم و برسی کنیم.

4. درست بودن کد نوشته شده

برای برسی کد نوشته شده متد خودکار اجرا شدن کد را پیاده کردیم طوری که بدون دخالت کاربر تمام بازی اجرا شود

با این کار سرعت برسی کد ها افزایش میابد و میتوان از اجرای درست بازی تا اخرین مرحله مطمئن بود.

5. ایجاد قوانین بهتر برای بازی

برای پیشرفت بازی و جلوگیری از گیر افتاد در حلقه بینهاید برای قسمت جواب دادن به یک کاربر از روش جدیدی استفاده میکنیم.

فرض کنید بازیکن شماره 1 سه کارت را انتخاب میکند (آشپزخانه و سوفیا و کشومخفی) و بازیکن شماره 2 جواب میدهد و آن کارت آشپزخانه را به بازیکن شماره 1 نشان میدهد

اگه بعد از چند دور مجدد بازیکن (آشپزخانه و سوفیا و کشومخفی) را انتخاب کرد بازیکن شماره 2 نمیتواند مجدد آشپزخانه را نشان بدهد یا باید کارت جدیدی نشان بدهد یا کنار کشیده و اجازه بدهد بازیکن بعدی جواب بدهد.

این قسمت به طور خودکار پیاده سازی شده و کاربر نمیتواند تصمیم بگیرد تا بازی پیش برود.