TDD Exam — Texas Hold’ em Poker Hand Comparison (Language Agnostic)

You have the afternoon to build a **complete** Texas Hold’ em hand evaluator and comparer using **TDD**. You may use any programming language and any test framework. You will have access to the internet.

Source of truth for hand categories and ordering:
https://en.wikipedia.org/wiki/List_of_poker_hands

---

1) Goal

Implement a program/module that, given:

- **5 community cards** (the board)
- and for each player **2 hole cards**

will:

1. Determine each player’ s **best 5-card poker hand** from their 7 available cards.
2. Compare all players and return the **winner(s)** (support ties / split pots).
3. Provide, for each player, the **chosen best 5 cards** (mandatory) and the recognized hand category.

This is a **hand comparison** exercise only (no betting logic).

---

2) Core rules (Texas Hold’ em)

- Each player can form a hand using:
  - both hole cards, or
  - one hole card, or
  - **zero** hole cards (“board plays”).
- The resulting hand is always the **best possible 5-card hand** out of the 7 cards.
- Ties are possible; if multiple players have the same best hand, the result is a **split**.

No suit-based tie-breaking: suits only matter to detect a flush. There is no rule like “spades beat hearts” .

---

3) Hand categories and global ordering (highest → lowest)

Use the standard order (per Wikipedia):

1. Straight flush (includes the special case often called “royal flush”)
2. Four of a kind
3. Full house
4. Flush
5. Straight
6. Three of a kind
7. Two pair
8. One pair
9. High card

(If you choose to represent “royal flush” separately, that’s fine, but it must behave like the best straight flush.)

---

4) Tie-break rules (when categories are equal)

When two hands have the same category, compare them using these rules in order. If all compared values are equal, it is a **tie**.

4.1 Straight / Straight flush

- Compare the **highest card** of the straight.
- Special case: (A,2,3,4,5) is a valid straight where **Ace is low** (this is a 5-high straight).
- (10,J,Q,K,A) is an Ace-high straight.
- No wrap-around straights (e.g., (Q,K,A,2,3) is invalid).

4.2 Four of a kind

- Compare the rank of the **four** cards (quads).
- If equal, compare the **kicker** (the remaining card).

4.3 Full house

- Compare the rank of the **three-of-a-kind** part first.
- If equal, compare the rank of the **pair** part.

4.4 Flush

- Compare the five flush cards in descending order:
  - highest flush card, then next highest, etc.
- If all five ranks match, tie.

4.5 Three of a kind

- Compare rank of the **triplet**.

- Then compare remaining two **kickers** in descending order.

4.6 Two pair

- Compare rank of the **higher pair**.
- Then compare rank of the **lower pair**.
- Then compare the **kicker**.

4.7 One pair

- Compare rank of the **pair**.
- Then compare the remaining **three kickers** in descending order.

4.8 High card

- Compare the five cards in descending order.

---

5) Mandatory output: chosen best 5 cards

Your evaluation output **must** include the **exact 5 cards** chosen as the player’ s best hand.

5.1 Requirements for `chosen5`

- Must contain **exactly 5 distinct cards**.
- Must be a subset of the player’ s available 7 cards (2 hole + 5 board).
- Must match the declared category and tie-break result.

5.2 Ordering requirement (to make tests deterministic)

Return the `chosen5` in a **consistent order** and document it in your README.
Recommended: order them by “importance” for the category (matching tie-break logic). Examples:

- Straight: highest-to-lowest in straight order (wheel as `5,4,3,2,A`)
- Four of a kind: four cards first (quad rank), then kicker
- Two pair: higher pair, lower pair, kicker
- Flush / High card: descending ranks

If multiple equally-valid `chosen5` exist (rare but possible), your implementation must pick one consistently.

---

6) Input validity assumptions

State (in README) whether you:

- assume there are **no duplicate cards**, or
- validate and reject invalid input.

Either approach is acceptable if it is clearly documented and tested accordingly.

---

7) Examples (for understanding; do not hardcode)

These are good candidates for tests. Suits are only shown where relevant.

Example A — Ace-low straight (wheel)

- Board: A♣ 2♦ 3♥ 4♠ 9♦
- Player: 5♣ K♦
  Best:
- category: Straight
- chosen5 (example ordering): 5♣ 4♠ 3♥ 2♦ A♣ (5-high)

Example B — Ace-high straight

- Board: 10♣ J♦ Q♥ K♠ 2♦
- Player: A♣ 3♦
  Best:
- category: Straight
- chosen5: A♣ K♠ Q♥ J♦ 10♣ (A-high)

Example C — Flush with more than 5 suited cards available

- Board: A♥ J♥ 9♥ 4♥ 2♣
- Player: 6♥ K♦
  Best:
- category: Flush
- chosen5 must be the best five hearts: A♥ J♥ 9♥ 6♥ 4♥

Example D — Board plays (tie)

- Board: 5♣ 6♦ 7♥ 8♠ 9♦
- Player1: A♣ A♦
- Player2: K♣ Q♦
  Best for both:
- category: Straight
- chosen5: 9♦ 8♠ 7♥ 6♦ 5♣
  Result: tie / split

Example E — Quads on board, kicker decides

- Board: 7♣ 7♦ 7♥ 7♠ 2♦
- Player1: A♣ K♣

- Player2: Q♣ J♣
  Best:
- both category: Four of a kind (7s)
- chosen5 includes four 7s + best kicker
- Player1 wins (A kicker)

---

8) TDD requirements

You are graded primarily on your **TDD process and test quality**, not on “finishing fastest”.

You must demonstrate:

1. Incremental development via tests (small steps).
2. Tests covering:
   - each hand category detection,
   - tie-break rules within each category,
   - best-of-7 selection (including “board plays”),
   - split/tie outcomes,
   - Ace-low straight edge case.
3. Refactoring performed safely:
   - You must be able to show **refactoring steps** that keep tests green (e.g., commit history where refactors are separate from feature changes).

---

9) Git requirement

Use Git throughout the exam. Your repository history must make it clear that you:

- wrote tests first for new behavior (or at least in small cycles),
- implemented to make them pass,
- refactored while staying green.

(How you structure commits is up to you, but your history should be readable and show your iterations.)

---

10) What is explicitly out of scope

- Betting rules, blinds/antes, turn order, side pots, chip stacks.
- Jokers or wildcards.
- Suit ranking for tie-breaks.

If anything is ambiguous, rely on the Wikipedia page above as the rules reference, and document your interpretation in the README. Or just send me a message.

Notation

Create a github (gitlab... works as well) repository and send it to me with the list of students in a students.txt file at the root of the git repository.