BREAKTHROUGH!

Programming Tasks

These questions require you to load the **Skeleton Program** and to make programming changes to it.

Note that any alternative or additional code changes that you deemed appropriate to make must also be evidenced – ensuring that it is clear where in the Skeleton Program those changes have been made.

Task 1 — **Difficulty:** Easy **Marks:** 2

This question refers to the **playGame** method of the **Breakthrough** class.

The number of cards left in the deck should be printed out after the current sequence and before the cards in the player's hand each turn.

Test the changes you have made:

Run the game and play two turns, showing the number of cards in the deck decreasing appropriately.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **playGame** method
- SCREEN CAPTURE(S) showing the required test

Task 2 **Difficulty:** Easy **Marks:** 5

This question refers to the **playGame** and **getChoice** methods of the **Breakthrough** class and the creation of a new attribute (with accessor methods), **peekUsed** in the **Lock** class.

Introduce a **(P)eek** option. This can be used once per lock, and allows a player to peek into the deck to see the next three upcoming cards. There should be a new command in **playGame** that only appears if the 'deck peek' is still available.

Create a new attribute in the **Lock** class called **peekUsed**. Create accessor methods to the **Lock** class to update and read the **peekUsed** attribute (get/set).

Update the **getChoice()** method in the **Breakthrough** class to give the user the option to '(P)eek'. This menu option should only appear if the **peekUsed** attribute is **false**.

Introduce an option to the menu in the **playGame()** method to accept 'P' as one of the menu choices. This menu option should only appear if the **peekUsed** attribute is **false**. Display the next three cards in the deck using the **getCardDescriptionAt()** method. Set the **peekUsed** attribute appropriately once the peek option has been chosen by the user.

When the player is given a new lock, set the **peekUsed** attribute appropriately to allow the user to use the peek option again.

Test the changes you have made:

Run the game and peek (peek is an option, it works and then it's no longer an option), peek again to make sure it doesn't work even though the option isn't displayed. Solve a lock and check that peek is now an option again.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **playGame** method
- PROGRAM SOURCE CODE showing changes made to the **getChoice** method
- PROGRAM SOURCE CODE for the new **peekUsed** attribute
- SCREEN CAPTURE(S) showing the required test.

Task 3 **Difficulty:** Easy **Marks:** 3

This question refers to the **playCardToSequence** method of the **Breakthrough** class.

Under the rules of the game, a player cannot play two cards of the same type sequentially. Currently there is no error message warning the player when they attempt to do this, however.

Modify the **playCardToSequence** method in the **Breakthrough** class to introduce an error message which tells the user that they cannot play two cards of the same type sequentially.

Use the **getCardDescriptionAt** method to highlight to the user which card they have just tried to play and explain that it is the same as the type just played.

Test the changes you have made:

Run the game and show at least one turn played where the error does not get shown and one where it shows the new error message under the correct conditions of playing a duplicate tool. Make sure to show that (1) the error message is displayed and (2) the card is not played or discarded.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **playCardToSequence** method
- SCREEN CAPTURE(S) showing the required test

Task 4 **Difficulty:** Easy **Marks:** 6

This question refers to the **playGame** and **getChoice** methods and the creation of a new private attribute, **mulliganUsed** of the **Breakthrough** class.

Each player gets 1 ‘mulligan’ per game where they can take all the cards in their hand, the deck, the discard pile and the sequence, put them together and shuffle up and deal again. Any difficulty cards drawn (when repopulating the player’s hand) should be sent to the discard pile. The player’s score and the current lock including any solved challenges will remain unchanged.

Create a new attribute in the **Breakthrough** class called **mulliganUsed** which is initialised to **false**. If the **mulliganUsed** is **false** then display an additional **(M)ulligan** option each turn for the player and once the mulligan has been used, set the **mulliganUsed** attribute to **true**, which should mean that the **(M)ulligan** option is no longer displayed or usable.

Test the changes you have made:

Run the game, solve one challenge, use mulligan, play one card to the sequence, choose M (in an attempt to mulligan again despite no menu option).

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **playGame** method
- PROGRAM SOURCE CODE showing changes made to the **Breakthrough** class
- PROGRAM SOURCE CODE showing changes made to the **getChoice** method
- SCREEN CAPTURE(S) showing the required test

Task 5 — **Difficulty:** Easy **Marks:** 3

This question refers to the **playGame** and **getChoice** methods of the **Breakthrough** class.

The player will have a new option in **playGame** to **(Q)uit**, and for this they will get 1 point added to their score for each card remaining in the deck. Print out their final score as they quit.

Note that the code should exit cleanly/nicely without using any **System.exit()** type functions (although break/continue are allowed of course).

Test the changes you have made:

Play one turn of a game, choose quit.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **playGame** method
- PROGRAM SOURCE CODE showing changes made to the **getChoice** method
- SCREEN CAPTURE(S) showing the required test

Task 6

Difficulty: Medium **Marks:** 5

This question refers to the **getCardFromDeck** method of the **Breakthrough** class and the creation of a new method, **displayStats**, modifying two existing methods, **addCard** and **removeCard**, as well as adding three new attributes, **numPicks**, **numFiles** and **numKeys**, in the **CardCollection** class.

Introduce a stats / card count to the **CardCollection** class which keeps track of which cards have come out of the deck and calculates the % chance that the next card tile in the deck is X type (or XYZ types).

Introduce three new attributes to the **CardCollection** class called **numPicks**, **numFiles** and **numKeys**, which will be updated every time a **ToolCard** is added to or removed from the **CardCollection**.

Create a new method in the **CardCollection** class called **displayStats**. This method should calculate the percentage chance of the next card being a key, pick or file based on the number of each card and the number of cards left in the deck.

When the player receives a difficulty card, use the **displayStats** method together with the **getNumberOfCards** method in the **CardCollection** class to display the following on the screen before they choose 'lose a key or discard 5 cards from the deck'.

There is a X% chance that the next card will be a key, a Y% chance that it will be a file and a Z% chance that it will be a pick.

The percentages should be displayed to two decimal places.

Replace X, Y and Z with the appropriate values. Note that they will not normally add up to 100% because there are also difficulty cards in the deck.

Test the changes you have made:

Run the game until a difficulty card is drawn and show the printout of the statistics in the correct place (after the hand and before asking which card).

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **getCardFromDeck** method
- PROGRAM SOURCE CODE showing changes made to the **CardCollection** class
- SCREEN CAPTURE(S) showing the required test

Task 7 **Difficulty:** Medium **Marks:** 6

This question involves the **createStandardDeck**, **processLockSolved** and **playCardToSequence** methods of the **Breakthrough** class, as well as the creation of a new **setCardToolkit** method in the **Card**, **ToolCard** and **CardCollection** classes.

Introduce three new 'multi-tool' cards – a **multi-pick (P m)**, a **multi-key (K m)** and a **multi-file (F m)**.

At the start of a standard game (not when loading a save game file), the deck should contain one of each of these new types of card. Multi-tool cards can be dealt to the player's hand in the same way as normal cards are.

On playing a multi-tool card, the player should be given the option to choose which toolkit they want to assign the card to before it is added to the sequence, therefore allowing a multi-tool card to be applied to any lock challenge of that type.

When a lock has been solved, three new multi-tool cards (one of each type) are added to the deck to be available for the next lock and the deck is reshuffled (as normal).

Test the changes you have made:

Play the game and show the use of at least one multi-tool card, the print screen must show the hand and sequence both before and after the multi-tool is played.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **createStandardDeck** method
- PROGRAM SOURCE CODE showing changes made to the **processLockSolved** method
- PROGRAM SOURCE CODE showing changes made to the **playCardToSequence** method
- PROGRAM SOURCE CODE for the new **setCardToolkit** method (in **Card**, **ToolCard** and **CardCollection** classes)
- SCREEN CAPTURE(S) showing the required test.

Task 8 **Difficulty:** Medium **Marks:** 6

This question refers to the **getLockDetails** method of the **Lock** class and **playGame** method of the **Breakthrough** class.

Challenges are to be marked as 'partially met' (rather than just 'met' or 'not met') if they are partially solved. A challenge is partially met if the end of the sequence (last one or two cards) matches the start of an unsolved challenge.

Modify the call to **getLockDetails** from **playGame** to pass in the sequence.

Modify **getLockDetails** so that if the challenge is not met then it checks to see whether it is partially met. For challenges of three cards, only check the last two cards and it becomes partially met if the last card of the sequence matches the first card of the challenge or the second last card of the sequence matches the first card of the challenge and the last card of the sequence matches the second card of the challenge.

In general, check N-1 cards where N is the number of cards in the challenge – meaning of course that challenges of one card cannot be partially met. You only need to solve the problem for challenges of three cards exactly.

Test the changes you have made:

Run the game and play one card to the sequence that doesn't match any of the challenges, then play one towards one of the three card challenges that matches the first card for that challenge, and print screen showing this entire turn.

Then play a second card to the sequence that matches the second card of the three card challenge.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **playGame** method
- PROGRAM SOURCE CODE showing changes made to the **getLockDetails** method
- SCREEN CAPTURE(S) showing the required test

Task 9 **Difficulty:** Medium **Marks:** 5

This question refers to the **playGame** method of the **Breakthrough** class.

Introduce a bonus for solving locks using fewer cards. Once the first card is played towards the sequence for a new lock, a counter starts and one is added every time a player makes a move (discarding or playing to the sequence).

Once a lock is solved (all the challenges), a player receives an extra point for every point under 20 on the counter, after which it is reset. The player simply receives 0 if the counter is 20 or more. Print out a message confirming the bonus points that were awarded (including 0 if that's the case).

Test the changes you have made:

Run the game and play two locks, one solved in under 20 cards to show a bonus score and one solved in over 20 cards to show a bonus score of 0.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **playGame** method
- SCREEN CAPTURE(S) showing the required test

Task 10 **Difficulty:** Medium **Marks:** 9

This question refers to the **processLockSolved**, **setupGame** and **getCardFromDeck** methods as well as the creation of a new method, **addGeniusCardToDeck** of the **Breakthrough** class and the creation of a new class called **GeniusCard**.

Introduce a new 'Genius Card' which is added to deck at the start of a lock. There should be a 25% chance of having a 'Genius Card' in a deck.

A player can choose to use the 'Genius Card' when they draw it to solve a challenge instantly (it should ask which challenge) or it will be discarded and then reshuffled into the deck as normal with all the cards from the discard pile.

Note that if a **GeniusCard** is drawn when filling up the hand it should be discarded automatically and a message should be printed to this effect.

Create a method called **addGeniusCardToDeck** which has a 25% chance of adding one **GeniusCard** to the deck. This should be called from **processLockSolved** and **setupGame**.

Create a new class for the **GeniusCard** which inherits **Card** with **cardType** equal to 'Gen' and modify the **getCardFromDeck** method of **Breakthough** to ensure that the card is processed correctly when drawn.

Test the changes you have made:

Run the game and play until a 'Genius Card' is drawn, then choose yes and select the last unsolved challenge in the current lock.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **processLockSolved** method
- PROGRAM SOURCE CODE showing changes made to the **setupGame** method
- PROGRAM SOURCE CODE showing changes made to the **getCardFromDeck** method
- PROGRAM SOURCE CODE for the new **GeniusCard** class
- PROGRAM SOURCE CODE for the new **addGeniusCardToDeck** method
- SCREEN CAPTURE(S) showing the required test

Task 11 **Difficulty:** Hard **Marks:** 12

This question refers to the addition of a new attribute in the **Breakthrough** class called **credits** and to the **getCardFromDeck** method of the **Breakthrough** class as well as the creation of a new method, **printToolsAvailable,** for the **CardCollection** class.

Introduce the concept of 'Buying a tool' from the deck.

Add a new attribute, **credits**, to the **Breakthrough** class which contains the number of credits the player currently has. At the start of the game, the player has 10 credits. When a player has played a card to the sequence or discarded a card, if they have at least 2 credits remaining, they should be asked if they would like to buy a tool (y/n) before their hand is refilled from the deck. If they choose 'n' then they simply draw a card as normal, but otherwise the new card will be the tool card that they purchased.

Players can 'buy' a 'Key' card at the cost of 3 credits, and 'file' or 'pick' cards at the cost of 2 credits each. When the player chooses 'y' to buy a tool, they should be prompted with the following menu (items which have 0 availability should not be listed).

```
1.    F a (1 available)
2.    F b (1 available)
3.    F c (1 available)
4.    P a (1 available)
5.    P b (1 available)
6.    P c (1 available)
7.    K a (1 available)
8.    K b (1 available)
9.    K c (1 available)
10.   No Tool (buy nothing)
```

***Note**: the actual number available should be given, not just 1*

***Note**: keys (items 7–9) should only be listed if the player has at least 3 credits left. All menu items should retain the numbers given above even if some menu items are missing, e.g. item 10 should always be the 'No Tool' option (effectively the player changed their mind).*

The new **printToolsAvailable** method should take one parameter, **keysAvailable**, which is **true** if the player has at least 3 credits and otherwise is **false**. It should return an array containing the index of the first available tool card of each type; for example, if the deck contains three files from toolkit a, the first of which is at index 3 in the deck, no files from toolkit b and one file from toolkit c which is at index 12 in the deck, then the first part of the array returned would look like this:

```
[ 3, -1, 12, …
```

***Note:** -1 is used to indicate that no tool is available in the deck*

Test the changes you have made:

1. Run the game and play any card to the sequence, then choose 'y' when asked if you would like to buy a tool. Select any tool listed as available, play it to the sequence and then choose 'y' again when asked if you would like to buy a tool; show all the output produced including both menus presented and the tool card being added to the player's hand each time.
2. Continue playing the game and buying tools until you have spent a total of 8 credits (4 picks/file or 1 pick/file and 2 keys) and then show the printed list of tools available when you next choose to buy a tool.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **getCardFromDeck** method
- PROGRAM SOURCE CODE for the new **credits** attribute
- PROGRAM SOURCE CODE for the new **printToolsAvailable** method
- SCREEN CAPTURE(S) showing the required test

Task 12 **Difficulty:** Hard **Marks:** 4

This question refers to **checkIfLockChallengeMet** method of the **Breakthrough** class.

Create an 'Advanced' mode where, for any challenge that requires three or more tool cards to solve, once the challenge is solved move the cards used to solve it from the sequence to the discard pile, exposing the previous card on the sequence, which could then possibly be used in solving another challenge.

For example, if the sequence contains:

```
Fa, Kc, Pb
```

and the current challenge is Pb, Kb, Fb. Suppose you play Kb and Fb to the sequence; this will solve the current challenge but instead of the sequence extending to:

```
Fa, Kc, Pb, Kb, Fb
```

it will be contracted to:

```
Fa, Kc
```

and the Pb, Kb and Fb cards from the challenge that was just solved will be added to the discard pile.

Test the changes you have made:

Run the game and restart until you get a Lock with at least one challenge of one card and one challenge of three cards. Play until you solve the single card challenge and then play until you solve a three card challenge. The screen capture(s) should show the Lock, Sequence and Hand before you play the final card to solve the three card challenge and the Lock and Sequence after you play it.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **checkIfLockChallengeMet** method
- SCREEN CAPTURE(S) showing the required test

Task 13 **Difficulty:** Hard **Marks:** 8

This question refers to **playGame** and **getChoice** methods and to the creation of a new **saveGame** method in the **Breakthrough** class. It also requires the creation of new **getChallengesAsString** and **getChallengesMetAsString** methods in the **Lock** class.

The **playGame** menu should have a **(S)ave** option which will save the game in its current state and allow it to be reloaded (from the main menu when you first start the game).

In order to understand the format of the save game file, you will have to inspect the **game1.txt** file and the **loadGame** method of the **Breakthrough** class.

Print out a suitable message stating whether the game was saved successfully or not.

Test the changes you have made:

1. Take a copy of the **game1.txt** file and rename it **backup.txt**.
2. Run the game until you get a lock with at least two challenges. Solve one challenge and then save the game as '**game1.txt**' (it shouldn't prompt you). Load the game and ensure that the state of it has been correctly restored.
3. Restore the original **game1.txt** from **backup.txt**.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **playGame** method
- PROGRAM SOURCE CODE showing changes made to the **getChoice** method
- PROGRAM SOURCE CODE for the new **saveGame** method
- PROGRAM SOURCE CODE for the new **getChallengesAsString** method
- PROGRAM SOURCE CODE for the new **getChallengesMetAsString** method
- SCREEN CAPTURE(S) showing the required test

Task 14 **Difficulty:** Hard **Marks:** 6

This question refers to **playGame** and **playCardToSequence** methods and the creation of a new attribute, **bonusPool**, in the **Breakthrough** class. It also requires the creation of a new public method, **isPartial**, in the **Lock** class, which takes **sequence** as a parameter.

Introduce a bonus for playing consecutive cards to a lock that solves a challenge. More specifically, each card played in a row that goes towards solving a challenge will add 5 to the bonus pool and that bonus pool will be added to the score for each card.

For example, the bonus pool is 0 and a player plays a card towards challenge 1: the bonus pool of 0 is added to their score along with their normal score and the bonus pool is increased to 5. If the player does anything except play another correct card towards challenge 1, then the bonus pool will be reset to 0; otherwise they will get the score for the card played as normal, plus the bonus pool of 5, and then the bonus pool will be increased to 10 and so on.

Test the changes you have made:

Run the game and keep discarding until you have all three cards required to solve a challenge, then solve it one card after another; continue playing and play a card to a challenge and then a card to the sequence that is not part of the challenge.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **playGame** method
- PROGRAM SOURCE CODE showing changes made to the **playCardToSequence** method
- PROGRAM SOURCE CODE for the new **bonusPool** attribute
- PROGRAM SOURCE CODE for the new **isPartial** method
- SCREEN CAPTURE(S) showing the required test

Task 15 — **Difficulty:** Extreme **Marks:** 10

This question refers to the **processLockSolved**, **getCardFromDeck** and **checkIfPlayerHasLost** methods, and to the creation of new private **generateSolubleLock** and **generateChallenge** methods and a new private attribute **finalLock** in the **Breakthrough** class. It also requires the creation of a new public **isSoluble** method in the **Lock** class that takes the **deck** and **hand** as parameters.

EXTRA FILE NEEDED: **game2.txt**

Every lock presented must be solvable based on the cards left in the deck, even if doing so would exhaust the deck. If the lock cannot be solved, then choose a new random lock. If this happens 10 times in a row (without a suitable lock being found) then display a message 'Final Lock' to the user and generate a lock with two challenges that can be solved.

Once those challenges are solved, there should be a message from **checkIfPlayerHasLost** that, instead of saying the player lost, prints out 'You have solved the final lock. Your final score is:' + **Score**.

When approaching this task you should ignore the effect of **Difficulty** cards and it is sufficient to simply check that the **deck** and **hand** combined contain the requisite number of each type of card for the next lock.

The attribute **finalLock** should be set to 0 at the start and then set to 1 in **processLockSolved** when the final lock is set. When **checkIfPlayerHasLost** runs, it should set **finalLock** to 2 if it is 1 (ensuring that the final turn is played). If **finalLock** is 1 and there are no cards left in the deck then the player doesn't lose until all the cards from their hand are gone.

Test the changes you have made:

1. Change the game to load the file **game2.txt** instead of **game1.txt** and then run the game and load game.
2. Play the game until the message 'Final Lock' is displayed, then solve that lock and print out the final turn.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **processLockSolved** method
- PROGRAM SOURCE CODE showing changes made to the **checkIfPlayerHasLost** method
- PROGRAM SOURCE CODE showing changes made to the **getCardFromDeck** method
- PROGRAM SOURCE CODE for the new **generateSolubleLock** method
- PROGRAM SOURCE CODE for the new **generateChallenge** method
- PROGRAM SOURCE CODE for the new **IsSoluable** method
- PROGRAM SOURCE CODE for the new **finalLock** attribute
- SCREEN CAPTURE(S) showing the required test