BREAKTHROUGH!

Skeleton Code Breakdown

Note: In the skeleton code released by AQA, all parameters are passed by value.

Class: Breakthrough

Identifier / Data		Description
<<constructor>> **Breakthrough**		
Parameters	n/a	Initialises several private attributes including • **deck** to a new **CardCollection** • **hand** to a new **CardCollection** • **sequence** to a new **CardCollection** • **discard** to a new **CardCollection** • **score** to 0 • **gameOver** to **False** • **locks** to an empty list • **currentLock** to an empty **Lock** • **lockSolved** to **False** Invokes the **LoadLocks()** method to load the external locks text file '**locks.txt**'.
Return values	n/a	
addDifficultyCardsToDeck (private)		
Parameters	n/a	Adds five **DifficultyCards** to the **deck**.
Return values	n/a	
checkIfLockChallengeMet (private)		
Parameters	n/a	Iterates through the **sequence CardCollection** concatenating together the string **sequenceAsString** with a comma and a space as the separator between each card description. As a new element from **sequence** is concatenated onto the end of **sequenceAsString**, the string is compared with the **Challenge** conditions using the **checkIfConditionMet()** method on the current lock to check whether a challenge has been met. This is tested incrementally because challenges can be different lengths. If a challenge has been met, **true** is returned, otherwise **false** is returned.
Return values	Boolean	
checkIfPlayerHasLost (private)		
Parameters	n/a	Checks to see if there are any cards left in the **deck**. If there are none, an appropriate message is displayed on the screen together with the final score; the game is over and the method returns **true**. If there are cards still left in the **deck**, the player has not lost yet, and **false** is returned, allowing the player to continue playing.
Return values	Boolean	
createStandardDeck (private)		
Parameters	n/a	Used by the **setupGame()** method to populate an empty **deck** with the correct **Files**, **Picks** and **Keys** for each toolkit. 5 **Picks** from toolkits a, b and c are added to the **deck** and then 3 **Files** and 3 **Keys** from toolkits a, b and c are also added.
Return values	n/a	

Identifier / Data		Description
getCardChoice (private)		
Parameters	n/a	Used by the **playGame()** method to ask the player which card in their **hand** they would like to use. Contains error handling to catch non-integer user input but does not catch data out of range.
Return values	**value** : Integer	
getCardFromDeck (private)		
Parameters	**cardChoice** : Integer	Used to get the next card from the **deck CardCollection** and add it to the **hand**. If the **deck CardCollection** has at least one card in it, the system will then check if the card at position zero in the **deck** is a **DifficultyCard**. If a **DifficultyCard** is found, the user is asked if they would like to lose a ‘Key’ card or discard the next 5 unseen cards from the **deck**. The **DifficultyCard** is then moved to the **discard CardCollection** and the **process()** method is invoked on the **DifficultyCard** passing the user’s **choice** as one of the parameters. The system then performs a check which occurs when repopulating the **hand** with cards following a card being played. If another **Difficulty** card is found during this process, the **Difficulty** card (or cards if there is more than one sequentially in the **deck**) is move automatically into the **discard CardCollection** rather than into the player’s **hand**. If the **deck** runs out of cards, the game ends.
Return values	n/a	
getChoice (private)		
Parameters	n/a	Used by the **playGame()** method to ask the player if they would like to use a card from their **hand** or display the current **discard CardCollection** on the screen.
Return values	**choice** : String	
getDiscardOrPlayChoice (private)		
Parameters	n/a	Used by the **playGame()** method to ask the player if they would like to play the selected card from their **hand** to the **sequence** or **discard** the selected card from their **hand** to the **discard CardCollection**.
Return values	**choice** : String	
getRandomLock (private)		
Parameters	n/a	Returns a randomly selected lock from the private attribute **locks**.
Return values	**lock**	
loadGame (private)		
Parameters	**fileName** : String	Uses the **fileName** parameter to load an external Game text file. Imports the current **score**, **challenges**, and **CardCollections** for the **hand**, **sequence**, **discard** and **deck**. **true** is returned if the file is loaded and processed correctly. If an error occurs, an error message is displayed and **false** is returned.
Return values	Boolean	

Identifier / Data		Description
LoadLocks (private)		
Parameters	n/a	Uses a hard-coded '**locks.txt**' file which contains the locks available for the game. Each line in the text file contains the challenges for a single lock. Each line from the file is split into a string list – **challenges**, using a semicolon as a delimiter. Each **Challenge** is then further split using a comma as a delimiter into the **conditions** for that challenge. The **conditions** are then added to a temporary **Lock** variable – **lockFromFile**, which is then added to the private attribute **locks** list for this game. If an error occurs, an error message is displayed to advise that the **locks.txt** file has not loaded correctly.
Return values	n/a	
moveCard (private)		
Parameters	**fromCollection** : CardCollection **toCollection** : CardCollection **cardNumber** : Integer	Moves a card at the position of **cardNumber** from the **CardCollection fromCollection** to the **CardCollection toCollection**. If the **fromCollection** is the player **hand** and the **toCollection** is the **sequence** and a valid card has been chosen (i.e. not out of range), the player's score is updated appropriately for the card being played. For all other moves from one collection to another, **score** is not updated. **score** is returned.
Return values	**score** : Integer	
playCardToSequence (private)		
Parameters	**cardChoice** : Integer	This method is used to move a card from the **hand** to the **sequence** to test it against a lock challenge. The system tests to see if the **sequence** has at least one card in the **CardCollection**. If it does, the system then checks to see if the card being played by the user is a different **toolType** as the previously played card. If the **toolTypes** do not match, the card can be played and the card is moved from the **hand** to the **sequence** and the **score** is updated appropriately for that card **toolType**. The system then gets a new card from the **deck** to put into the **hand**. If the **sequence** does not currently have any cards in it, the system moves the chosen card to the **sequence** and the **score** is updated appropriately. The system then uses the **checkIfLockChallengeMet()** method to confirm if the new card added to the **sequence** allows a **Challenge** to be met and if so displays an appropriate message on the screen and increases the player **score** by 5 points.
Return values	n/a	

Identifier / Data		**Description**
playGame (public)		
Parameters	n/a	This contains the main game loop. Checks to confirm if the private list attribute **locks** contains any locks loaded by the **LoadLocks()** method. If none have been loaded an error is displayed on the screen and the program quits. If the list does contain locks, it initialises the following private attributes: • **lockSolved** to **false** • Invokes the **setupGame()** method to set up the game The main game loop runs while the private attribute of **gameOver** is **false**. There is then an inner loop which runs while **gameOver** is **false** and the private attribute **lockSolved** is also **false**. The inner game loop displays the current user score, the conditions of the current lock and the contents of the **hand**, and **sequence CardCollections**. Using the **getChoice()** method to display a choice menu to the user, the game loop then uses selection to either display the **discard CardCollection** or use a card in the game. If the user selects to use a card, the system uses the **getCardChoice()** method to select a card. It then uses the **getDiscardOrPlayChoice()** method to confirm if the user wants to play or discard the chosen card. If the user selects discard, the system moves the selected card from the **hand** to the **discard CardCollection** and gets a new card from the **deck** using **getCardFromDeck()**. If the user selects play, the system uses the **playCardToSequence()** method to move the chosen card from the **hand** to the **sequence CardCollection**. Once a card has been played or discarded, the main game loop uses the **getLockSolved()** method on the **currentLock** to test to see if all the lock challenges have been met. If they have, the **lockSolved** attribute is set to **true** and a new lock is generated. If a lock has been solved, the inner loop returns back to the main game loop which checks if the game is over by invoking the **checkIfPlayerHasLost()** method. If this returns **true** the game ends.
Return values	n/a	
processLockSolved (private)		
Parameters	n/a	Increments the **score** by 10 and displays the user score on the screen. Uses an indefinite loop to iterate through the **discard CardCollection** returning all of the cards back to the **deck**. Reshuffles the **deck** using the **shuffle()** method and assigns a new lock using the **getRandomLock()** method with the private attribute **currentLock**.
Return values	n/a	

Identifier / Data		Description
setupCardCollectionFromGameFile (private)		
Parameters	**lineFromFile** : String **cardCol** : CardCollection	Used for processing lines 4 to 7 of the external save game file which are for processing the contents of **CardCollections** (namely the **deck**, **discard**, **hand** and **sequence**). Receives a single line of text (using the **lineFromFile** parameter) from the external game file as it is imported and processes it into a **CardCollection**. If the received **lineFromFile** contains text, it is split into a list of strings – **splitLine**, using the comma as the delimiter. The **splitLine** list is then processed iteratively to identify the card number and card type in each element and add it to a **CardCollection**. If a **DifficultyCard** is found, that is added instead of a normal **ToolCard**.
Return values	n/a	
setupGame (private)		
Parameters	n/a	Called from the **playGame()** method, this displays the first message of the game on the screen, asking if the player would like to load in an external game file or play a new game. If the player chooses to load the external file the system attempts to load the file '**game1.txt**'. If the file cannot be loaded the game quits. If the player chooses to play a new game, the system generates a new **deck** using the **createStandardDeck()** method and then shuffles it by invoking the **shuffle()** method. It then moves 5 cards from the **deck** to the **hand** to start the player off. The system then invokes the **addDifficultyCardsToDeck()** method to add 5 **DifficultyCards** into the **deck** and then reshuffles it again to ensure they are in random locations. The system then assigns a new lock at random to the private attribute **currentLock** using **getRandomLock()**.
Return values	n/a	
setupLock (private)		
Parameters	**line1** : String **line2** : String	Used for processing lines 2 and 3 of the external save game file which contain the challenges for the lock. The parameter **line1** contains line 2 from the external file and the parameter **line2** contains line 3 of the external file. Each line is split into a string list using a semicolon as the delimiter. The **line1** parameter is then further split using a comma as the delimiter to add a new challenge to the **currentLock**. A single line may contain multiple challenges. The **line2** parameter is split using a semicolon as the delimiter to populate the **met** status for each challenge using the **setChallengesMet()** method.
Return values	n/a	

Class: Challenge

Identifier / Data		Description
<<constructor>> **Challenge**		
Parameters	n/a	Initialises the following protected attributes: • **met** to **false** • **conditions** to an empty list
Return values	n/a	
getCondition (public)		
Parameters	n/a	Returns a list of strings of the **conditions** for this challenge in the lock.
Return values	**condition** : List (String)	
getMet (public)		
Parameters	n/a	Returns the value of the protected attribute: **met**.
Return values	**met** : Boolean	
setCondition (public)		
Parameters	**newCondition** : List (String)	Sets the value of the protected string list attribute: **condition** from the parameter **newCondition**.
Return values	n/a	
SetMet (public)		
Parameters	**newValue** : Boolean	Sets the value of the protected attribute: **met** from the parameter **newValue**.
Return values	n/a	

Class: Lock

This class does not have a specific constructor and therefore uses the default constructor

Identifier / Data		Description
addChallenge (public)		
Parameters	**condition** : List (String)	Initialises a new challenge and sets the value of its condition from the parameter **condition.** Appends the new challenge to the **challenges** protected attribute.
Return values	n/a	
checkIfConditionMet (public)		
Parameters	**sequence** : String	Returns **true** and sets the challenge to **met** by calling **SetMet()** if the **sequence** matches any unsolved challenge, otherwise it returns **false**.
Return values	Boolean	
convertConditionToString (private)		
Parameters	**c** : List (String)	Converts list of conditions into a single string for displaying on the screen by iterating through the parameter **c**, concatenating together a string **conditionAsString()** using a comma and a space as the delimiter.
Return values	**conditionAsString** : String	
getChallengeMet (public)		
Parameters	**pos** : Integer	Returns the **met** status of a **Challenge** at the position of **pos** in the **challenges** list.
Return values	Boolean	

Identifier / Data		Description
getLockDetails (public)		
Parameters	n/a	Used for displaying a challenge's current status by iterating through the **challenges** protected attribute, concatenating together the output string **lockDetails** which contains a string version of all the challenges for the lock and whether each has been met or not.
Return values	**lockDetails**: String	
getLockSolved (public)		
Parameters	n/a	Returns the status showing if a lock has been solved by iterating through the **challenges** protected attribute and returning **false** if there are any unmet ones, otherwise it returns **true**.
Return values	Boolean	
getNumberOfChallenges (public)		
Parameters	n/a	Returns the number of **Challenges** in the **challenges** List (the number of challenges in this lock).
Return values	Integer	
setChallengeMet (public)		
Parameters	**pos** : Integer **value** : Boolean	Uses the **SetMet()** method in the **Challenge** class to set the **met** attribute of a challenge at the position of **pos** in the **challenges** list to **met** or not **met** using the **value** parameter.
Return values	n/a	

Class: Card

Identifier / Data		Description
<<constructor>> **Card**		
Parameters	n/a	Initialises the **cardNumber** protected attribute using the static attribute (class variable) **nextCardNumber**. It then increments the static attribute (class variable) **nextCardNumber** which means that it will be the same and updated for all objects of this class. Initialises the **score** protected attribute to 0.
Return values	n/a	
getCardNumber (public)		
Parameters	n/a	Returns the value of the protected attribute **cardNumber**.
Return values	**cardNumber** : Integer	
getDescription (public)		
Parameters	n/a	Returns the protected attribute **cardNumber** casted as a string.
Return values	**cardNumber**: String	
getScore (public)		
Parameters	n/a	Returns the protected attribute **score**.
Return values	**score** : Integer	
process (public)		
Parameters	**deck** : CardCollection **discard** : CardCollection **hand** : CardCollection **sequence** : CardCollection **currentLock** : Lock **choice** : String **cardChoice** : Integer	Base class method for the **process()** method in derived classes to override.
Return values	n/a	

Class: ToolCard (inherits from Card)

Identifier / Data		Description
<<constructor>> **ToolCard**		
Parameters	**t** : String **k** : String **cardNo** : Integer	Initialises the following protected attributes: • **toolType** from parameter **t** • **kit** from parameter **k** • **cardNumber** from parameter **cardNo** Invokes the **setScore()** method to assign the correct score in the base class for the **toolType**.
Return values	n/a	
<<constructor>> **ToolCard**		
Parameters	**t** : String **k** : String	Initialises the following protected attributes: • **toolType** from parameter **t** • **kit** from parameter **k** Initialise **cardNumber** by calling the parent constructor. Invokes the **setScore()** method to assign the correct score in the base class for the **toolType**.
Return values	n/a	
getDescription (public)		
Parameters	n/a	Overrides the **getDescription()** method from the base class to return a concatenated string of the **toolType**, a space and the **kit** for this **ToolCard**
Return values	String	
setScore (public)		
Parameters	n/a	Assigns the correct **score** from the protected attribute **toolType**.
Return values	n/a	

Class: DifficultyCard (inherits from Card)

Identifier / Data		Description
<<constructor>> **DifficultyCard**		
Parameters	n/a	Initialises the protected attribute **cardType** to 'Dif'. Initialises **cardNumber** by calling the parent constructor.
Return values	n/a	
<<constructor>> **DifficultyCard**		
Parameters	**cardNo** : Integer	Initialises the protected attribute **cardType** to 'Dif'. Initialises **cardNumber** from parameter **cardNo**.
Return values	n/a	
getDescription (public) <<override>>		
Parameters	n/a	Overrides the **getDescription()** method from the base class to return the protected attribute **cardType**.
Return values	String	

Identifier / Data		Description
process (public) <<override>>		
Parameters	**deck** : CardCollection **discard** : CardCollection **hand** : CardCollection **sequence** : CardCollection **currentLock** : Lock **choice** : String **cardChoice** : Integer	Overrides the **process()** method from the base class to process the user choices from a difficulty card. When the user receives a difficulty card they are asked if they would like to discard a key or 5 cards from the deck. On choosing the option to discard a key, they are asked to select a key. This method then confirms if the choice parameter is valid. ***Although there are potential logic errors in this check, AQA have confirmed that the code is written as it was intended.*** If the **choice** parameter contains the position (it will be converted to an index by subtracting 1 from the position of a 'key' **ToolCard** in the player's **hand**, the card is removed from the **hand** and placed in the **discard CardCollection**. If the **choice** parameter does not point to a key (either through deliberate user choice or a logic error), 5 cards are removed from the **deck** and placed in the **discard CardCollection**.
Return values	n/a	

Class: CardCollection

Identifier / Data		Description
<<constructor>> **CardCollection**		
Parameters	**n** : String	Initialises the following protected attributes: • **name** from parameter **n** • **cards** to an empty list
Return values	n/a	
getCardDescriptionAt (public)		
Parameters	**x** : Integer	Returns a string containing the description of the **Card** at index **x** in the **cards** list by invoking the overridden **getDescription()** method in **Card**.
Return values	String	
getCardNumberAt (public)		
Parameters	**x** : Integer	Returns the **cardNumber** attribute of a **Card** at the index **x** in the **cards** list.
Return values	Integer	
getName (public)		
Parameters	n/a	Returns the value of the protected attribute **name**.
Return values	**name** : String	
addCard (public)		
Parameters	**c** (Card)	Appends the value of parameter **c** to the protected list attribute **cards**.
Return values	n/a	

Identifier / Data		**Description**
createLineOfDashes (private)		
Parameters	**size** : Integer	Used in formatting a **CardCollection** display UI.
Return values	**lineOfDashes** : String	Returns an appropriately sized **lineOfDashes** for the number of elements in a **CardCollection** or fixed at 10 if the **CardCollection** is greater than that (defined by parameter **size**).
getCardDisplay (public)		
Parameters	n/a	Used in formatting a **CardCollection** display UI. Creates the display output of a **CardCollection** by concatenating together the collection **name** and card descriptions from the protected list attribute **cards**. If there are no cards in the list, the collection name and 'empty' is returned. If there are cards in the collection, a list of dashes is created which is either appropriately sized for the number of cards in the collection or is fixed at 10 if the number of cards in the collection is greater than 10. This is to ensure that the display fits correctly in the terminal window. It then uses indefinite iteration to loop through the **cards** list using the **getDescription()** method to get a string description of the card at each element and concatenate it with a space and the **|** (pipe) symbol to create a visual 'line of cards'. It then creates a second line of dashes to concatenate underneath the 'line of cards' and returns the completed output.
Return values	**cardDisplay** : String	
getNumberOfCards (public)		
Parameters	n/a	Returns the number of cards in the protected list attribute **cards**.
Return values	Integer	
removeCard (public)		
Parameters	**cardNumber** : Integer	Returns the card from **cards** list at the index **cardNumber** and removes it from **cards**. If **cardNumber** is not a valid index, **null** is returned.
Return values	**cardtoGet** : Card	
shuffle (public)		
Parameters	n/a	Uses definite iteration to perform 10000 movements of cards from one random position to another in the protected list attribute **cards** in order to generate a pseudo random shuffle.
Return values	n/a	