6.	Define an abstract class 'Themepark' and inherit 2 classes 'Queensland' and 'Veegaland' from the abstract class. In both the theme parks, the entrance fee for adults is Rs. 500 and for children it is Rs. 300. If a family buys 'n' adult tickets and 'm' children tickets, define a method in the abstract class to calculate the total cost. Also, declare an abstract method playGame() which must be redefined in the subclasses. In Queensland, there are a total of 30 games. Hence create a Boolean array named 'Games' of size 30 which initially stores false values for all the elements. If the player enters any game code that has already been played, a warning message should be displayed and the user should be asked for another choice. In Veegaland, there are a total of 40 different games. Thus create an integer array with 40 elements. Here, the games can be replayed, until the user wants to quit. Finally display the total count of games that were repeated and count of the games which were not played at all. Low Level: Implement the game play for a single person Medium Level: Simulate the game play for 'n'+'m' family members at the theme park of their choice. High Level: Make necessary modifications in your code to have the top level structure 'ThemePark' defined as an interface.
7.	Read the Register Number and Mobile Number of a student. If the Register Number does not contain exactly 9 characters or if the Mobile Number does not contain exactly 10 characters, throw an IllegalArgumentException. If the Mobile Number contains any character other than a digit, raise a NumberFormatException. If the Register Number contains any character other than digits and alphabets, throw a NoSuchElementException. If they are valid, print the message 'valid' else 'invalid' Low Level: Implement the code to perform the above said operations Medium Level: Derive 3 different user-defined exceptions from Exception class and throw them instead of the given built-in exceptions. Provide appropriate code to handle them. High Level: Write the code for checking the mobile number in a different method and let the exception thrown in that method, be handled in the calling method. Design the try-catch blocks in such a way that the catch block for handling IllegalArgumentException is written only once.
8.	Within the package named 'primespackage', define a class Primes which includes a method checkForPrime() for checking if the given number is prime or not. Define another class named TwinPrimes outside of this package which will display all the pairs of prime numbers whose difference is 2. (Eg, within the range 1 to 10, all possible twin prime numbers are (3,5), (5,7)). The TwinPrimes class should make use of the checkForPrime() method in the Primes class. Low Level: Perform the above said operations. Medium Level: Create a sub-package 'mersennepackage' for the primespackage and include a class that checks if a given number is a Mersenne prime number or not. [A Mersenne prime is a prime number that is one less than a power of two. That is, it is a prime number of the form $M_n = 2^n - 1$, for some integer n. The exponents *n* which give Mersenne primes are 2, 3, 5, 7, 13, 17, 19, 31, ... and the resulting Mersenne primes are 3, 7, 31, 127, 8191, 131071, 524287, 2147483647, ...] High Level: Define a package 'specialnumbers' with a class named 'MersenneNumbers' that includes the method displayAllMersennePrime() to display all the Mersenne numbers within a given range.
9.	Define a class 'Donor' to store the below mentioned details of a blood donor. - Name, age, Address, Contactnumber, bloodgroup, date of last donation. Create 'n' objects of this class for all the regular donors at Vellore. Write these objects to a file. Read these objects from the file and display only those donors' details whose blood group is 'A+ve' and had not donated for the recent six months. Low Level: Implement the above problem.

	Medium Level: Include a method that modifies the date of last donation in the file for a Donor immediately after he/she donates blood. High Level: Assume there are 2 blood banks functioning at Vellore. Both of them maintain the details of their donors. Create a single repository of donors by storing the details of all the donors in the 2 files but without any redundancy. If name and contact number are the same for 2 donor objects in the two files, that donor detail should be included only once.
10.	Three students A, B and C of B.Tech-IT II year contest for the PR election. With the total strength of 240 students in II year, simulate the vote casting by generating 240 random numbers (1 for student A, 2 for B and 3 for C) and store them in an array. Create four threads to equally share the task of counting the number of votes cast for all the three candidates. Use synchronized method or synchronized block to update the three count variables. The main thread should receive the final vote count for all three contestants and hence decide the PR based on the values received. Low Level: Let the threads maintain their own set of 3 count variables and return them to the main() method where the individual counts sent by the threads are added and displayed. Medium Level: Implement the above mentioned scenario. High Level: Create an array of 'n' Thread objects and distribute the task of voting evenly among the threads. Also the threads have to fetch the vote details from a file.