Experiment No: 08

● **Aim: Write test cases for White box testing.**

● **Theory:**

● **Project Name: - The QR CODE SCANNER**

White Box Testing:

White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.

The white box testing contains various tests, which are as follows:

- Path testing
- Loop testing
- Condition testing
- Testing based on the memory perspective
- Test performance of the program.

Techniques Used in White Box Testing:

- Data Flow Testing
- Control Flow Testing
- Branch Testing
- Statement Testing
- Decision Testing

Data Flow Testing:

Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.

Control Flow Testing:

Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. Test cases represented by the control graph of the program.

Branch coverage technique:

Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.

Statement coverage technique:

Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code, out of total statements present in the source code.

Decision Testing:

This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Test Case 1:

This case the logo screen flash done move next page

```
  new Handler().postDelayed(new Runnable() {

// Using handler with postDelayed called runnable run method

  @Override

      public void run() {

      Intent i = new Intent(MainSplashScreen.this, FirstScreen.class);

       startActivity(i);

 // close this activity

       finish();
```

```
        }


    }, 5*1000); // wait for 5 seconds
```

Test Case 2: Login authentication+add databsae

1. private void initView() {
2. mCallbacks = new PhoneAuthProvider.OnVerificationStateChangedCallbacks() {

3. @Override
4. public void onCodeAutoRetrievalTimeOut(String verificationId) {
    a. if (progressDialog != null) {
    b. dismissProgressDialog(progressDialog);
    c. }
    d. notifyUserAndRetry("Your Phone Number Verification is failed.Retry again!");
5. }

6. @Override
7. public void onVerificationCompleted(PhoneAuthCredential credential) {
    a. Log.d("onVerificationCompleted", "onVerificationCompleted:" + credential);
    b. if (progressDialog != null) {
    c. dismissProgressDialog(progressDialog);
    d. }
    e. signInWithPhoneAuthCredential(credential);
8. }

9. @Override
10. public void onVerificationFailed(FirebaseException e) {
    a. Log.w("onVerificationFailed", "onVerificationFailed", e);

    b. if (progressDialog != null) {
    c. dismissProgressDialog(progressDialog);
    d. }

    e. if (e instanceof FirebaseAuthInvalidCredentialsException) {
    f. Log.e("Exception:", "FirebaseAuthInvalidCredentialsException" + e);
    g. } else if (e instanceof FirebaseTooManyRequestsException) {
    h. Log.e("Exception:", "FirebaseTooManyRequestsException" + e);

    i. }

```
        j. notifyUserAndRetry("Your Phone Number Verification is failed.Retry
           again!");
11. }

12. @Override
13. public void onCodeSent(String verificationId,
                i. PhoneAuthProvider.ForceResendingToken token) {
        b. Log.d("onCodeSent", "onCodeSent:" + verificationId);
        c. Log.i("Verification code:", verificationId);
14. }
15.         };
```

Test case 03 : scan QR code:

```
1.  public class ScannedBarcodeActivity extends AppCompatActivity {
2.
3.
4.      SurfaceView surfaceView;
5.      TextView txtBarcodeValue;
6.      private BarcodeDetector barcodeDetector;
7.      private CameraSource cameraSource;
8.      private static final int REQUEST_CAMERA_PERMISSION = 201;
9.      Button btnAction;
10.     String intentData = "";
11.     boolean isEmail = false;
12.
13.     @Override
14.     protected void onCreate(Bundle savedInstanceState) {
15.         super.onCreate(savedInstanceState);
16.         setContentView(R.layout.activity_scanned_barcode);
17.         initViews();
18.     }
19.
20.     private void initViews() {
21.         txtBarcodeValue = findViewById(R.id.txtBarcodeValue);
22.         surfaceView = findViewById(R.id.surfaceView);
```

```
23.       btnAction = findViewById(R.id.btnAction);
24.       btnAction.setOnClickListener(new View.OnClickListener() {
25.          @Override
26.          public void onClick(View v) {
27.             if (intentData.length() > 0) {
28.                if (isEmail)
29.                   startActivity(new Intent(ScannedBarcodeActivity.this, EmailActi
    vity.class).putExtra("email_address", intentData));
30.                else {
31.                   startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse(intentD
    ata)));
32.                }
33.             }
34.          }
35.       });
36.    }
37.
38.    private void initialiseDetectorsAndSources() {
39.
40.       Toast.makeText(getApplicationContext(), "Barcode scanner started", Toast
    .LENGTH_SHORT).show();
41.       barcodeDetector = new BarcodeDetector.Builder(this)
42.             .setBarcodeFormats(Barcode.ALL_FORMATS)
43.             .build();
44.
45.       cameraSource = new CameraSource.Builder(this, barcodeDetector)
46.             .setRequestedPreviewSize(1920, 1080)
47.             .setAutoFocusEnabled(true) //you should add this feature
48.             .build();
49.
50.       surfaceView.getHolder().addCallback(new SurfaceHolder.Callback() {
51.          @Override
52.          public void surfaceCreated(SurfaceHolder holder) {
53.             try {
```

```
54.                 if (ActivityCompat.checkSelfPermission(ScannedBarcodeActivity.t
    his, Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTE
    D) {
55.                     cameraSource.start(surfaceView.getHolder());
56.                 } else {
57.                     ActivityCompat.requestPermissions(ScannedBarcodeActivity.thi
    s, new
58.                             String[]{Manifest.permission.CAMERA}, REQUEST_CAMERA
    _PERMISSION);
59.                 }
60.
61.             } catch (IOException e) {
62.                 e.printStackTrace();
63.             }
64.         }
65.
66.         @Override
67.         public void surfaceChanged(SurfaceHolder holder, int format, int widt
    h, int height) {
68.         }
69.
70.         @Override
71.         public void surfaceDestroyed(SurfaceHolder holder) {
72.             cameraSource.stop();
73.         }
74.     });
75.
76.
77.     barcodeDetector.setProcessor(new Detector.Processor<Barcode>() {
78.         @Override
79.         public void release() {
80.             Toast.makeText(getApplicationContext(), "To prevent memory leaks
    barcode scanner has been stopped", Toast.LENGTH_SHORT).show();
81.         }
```

```
82.
83.         @Override
84.         public void receiveDetections(Detector.Detections<Barcode> detectio
    ns) {
85.             final SparseArray<Barcode> barcodes = detections.getDetectedIte
    ms();
86.             if (barcodes.size() != 0) {
87.                 txtBarcodeValue.post(new Runnable() {
88.                     @Override
89.                     public void run() {
90.
91.                         if (barcodes.valueAt(0).email != null) {
92.                             txtBarcodeValue.removeCallbacks(null);
93.                             intentData = barcodes.valueAt(0).email.address;
94.                             txtBarcodeValue.setText(intentData);
95.                             isEmail = true;
96.                             btnAction.setText("ADD CONTENT TO THE MAIL");
97.                         } else {
98.                             isEmail = false;
99.                             btnAction.setText("LAUNCH URL");
100.                                intentData = barcodes.valueAt(0).displayValue;
101.                                txtBarcodeValue.setText(intentData);
102.                            }
103.                        }
104.                    });
105.                }
106.            }
107.        });
108.    }
109.
110.
111.    @Override
112.    protected void onPause() {
113.        super.onPause();
```

```
114.            cameraSource.release();
115.        }
116.
117.        @Override
118.        protected void onResume() {
119.            super.onResume();
120.            initialiseDetectorsAndSources();
121.        }
122.    }
```

Advantages of White box testing:

- White box testing optimizes code so hidden errors can be identified.
- Test cases of white box testing can be easily automated.
- This testing is more thorough than other testing approaches as it covers all code paths.
- It can be started in the SDLC phase even without GUI.

Disadvantages of White box testing:

- White box testing is too much time consuming when it comes to large-scale programming applications.
- White box testing is much expensive and complex.
- It can lead to production error because it is not detailed by the developers.
- White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.

● Conclusion: -

White box testing for security is useful and effective. It should follow a risk-based approach to balance the testing effort with consequences of software failure. Architectural and design-level risk analysis provide the right context to plan and perform white box testing. White box testing can be used with black box testing to improve overall test effectiveness. It uncovers programming and implementation errors.