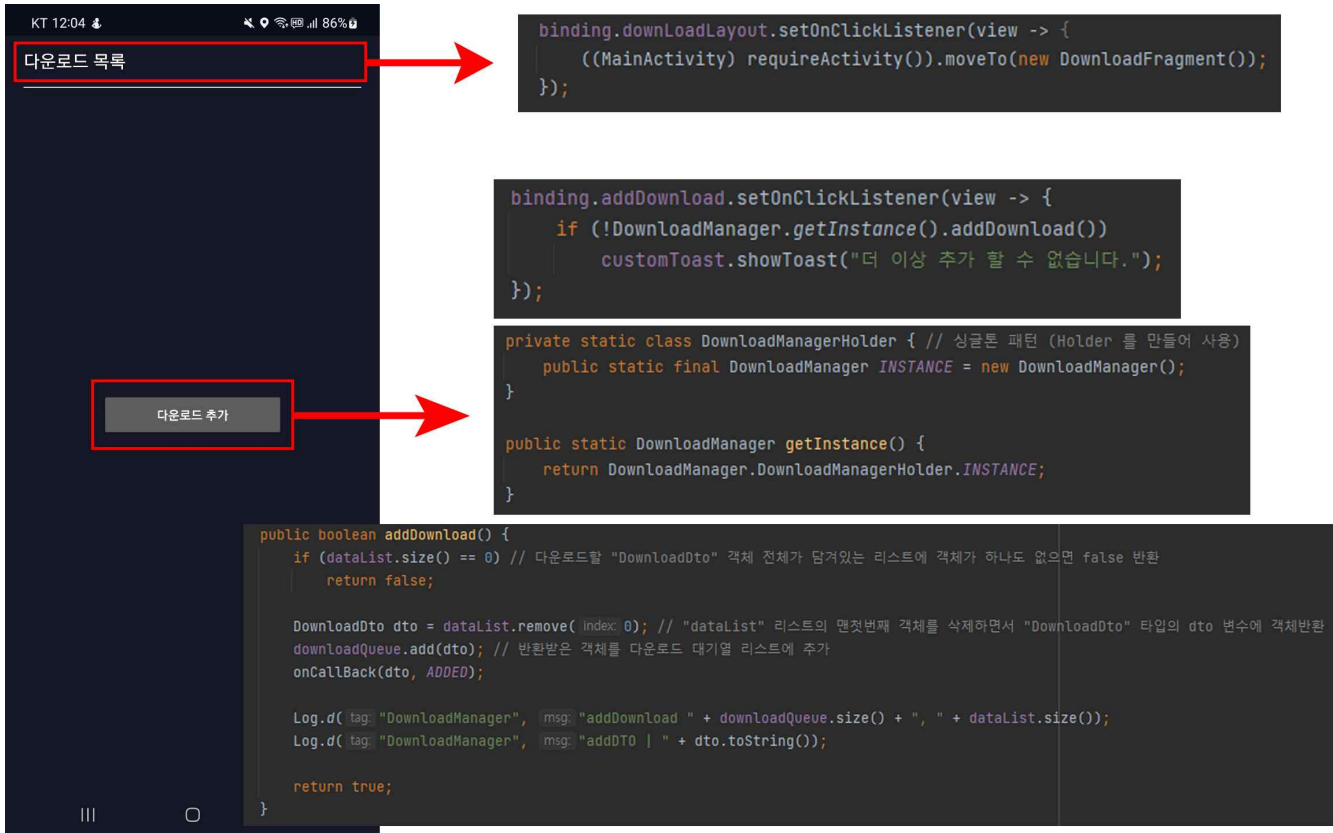


(주)파인원커뮤니케이션즈 장기 과제
LGU+DIVE 앱 시청콘텐츠 다운로드 기능구현 과제

1. 다운로드 목록 메인화면 (MainFragment Class)



The screenshot shows the MainActivity of the MainFragment Class. The '다운로드 목록' (Download List) button is highlighted with a red box, and a red arrow points to the following Kotlin code block:

```
binding.downloadLayout.setOnClickListener(view -> {
    ((MainActivity) requireActivity()).moveTo(new DownloadFragment());
});
```

The '다운로드 추가' (Add Download) button is also highlighted with a red box, and a red arrow points to the following Kotlin code block:

```
binding.addDownload.setOnClickListener(view -> {
    if (!DownloadManager.getInstance().addDownload())
        customToast.showToast("더 이상 추가 할 수 없습니다.");
});
```

Below these, the DownloadManagerHolder class is defined:

```
private static class DownloadManagerHolder { // 싱글톤 패턴 (Holder 를 만들어 사용)
    public static final DownloadManager INSTANCE = new DownloadManager();
}

public static DownloadManager getInstance() {
    return DownloadManager.DownloadManagerHolder.INSTANCE;
}
```

Finally, the addDownload() method is shown in detail:

```
public boolean addDownload() {
    if (dataList.size() == 0) // 다운로드할 "DownloadDto" 객체 전체가 담겨있는 리스트에 객체가 하나도 없으면 false 반환
        return false;

    DownloadDto dto = dataList.remove( index 0); // "dataList" 리스트의 맨첫번째 객체를 삭제하면서 "DownloadDto" 타입의 dto 변수에 객체반환
    downloadQueue.add(dto); // 반환받은 객체를 다운로드 대기열 리스트에 추가
    onCallBack(dto, ADDED);

    Log.d( tag: "DownloadManager", msg: "addDownload " + downloadQueue.size() + ", " + dataList.size());
    Log.d( tag: "DownloadManager", msg: "addDTO | " + dto.toString());

    return true;
}
```

*다운로드 추가 버튼 클릭시 DownloadManger 클래스에 있는 addDownload() 메소드 호출 ->
ArrayList<DownloadDto> dataList 리스트에 있는 DownloadDto 콘텐츠를 하나씩 LinkedList<DownloadDto> downloadQueue 리스트에 추가

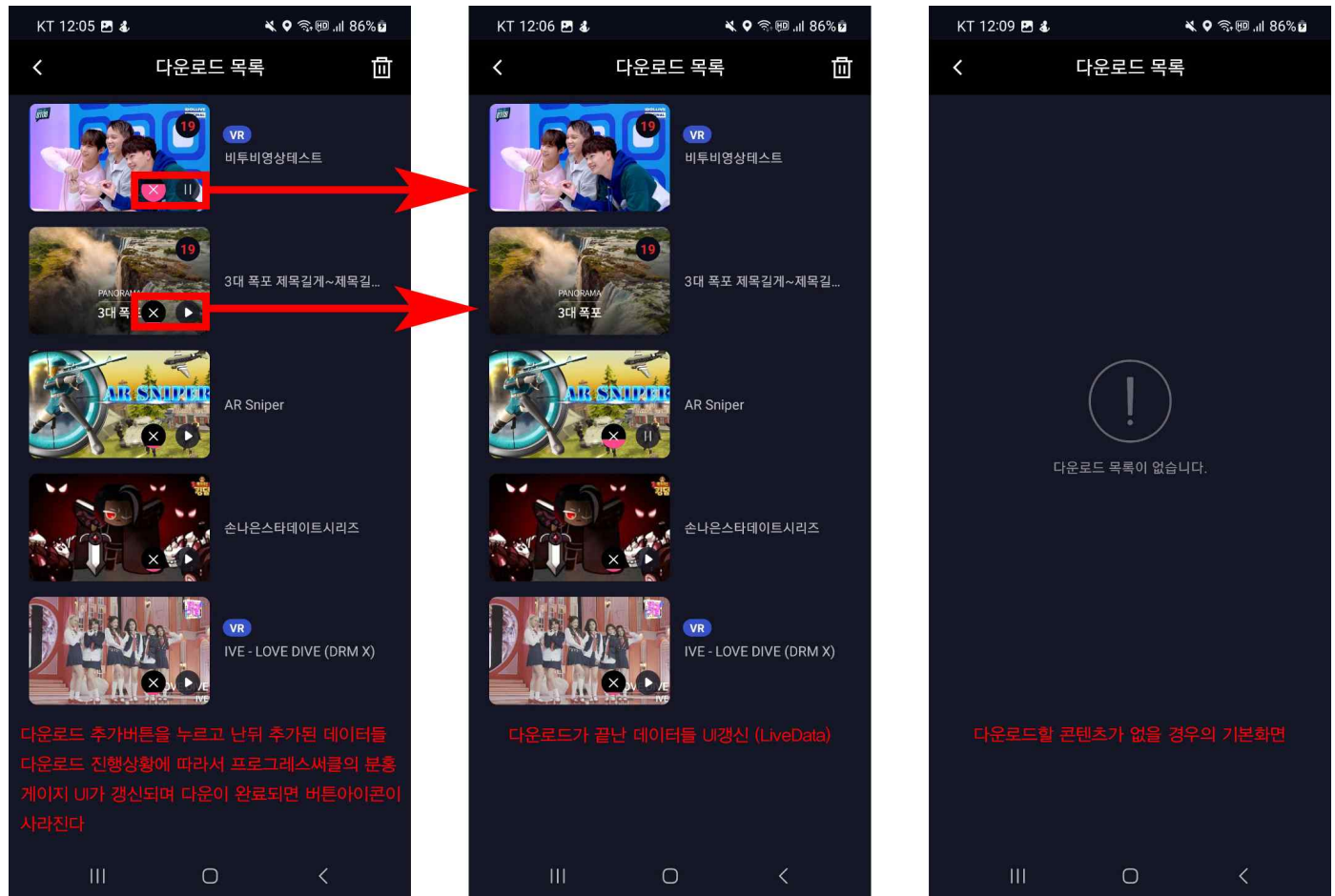
- DownloadDto Class (DataClass)

```
public class DownloadDto {

    private String contId; // 콘텐츠 ID
    private String ty1Code; // 콘텐츠 타입 코드 (AR : AR 콘텐츠, VR : VR 콘텐츠, LB : 라이브 콘텐츠)
    private String contNm; // 콘텐츠 이름
    private boolean isAdultCont; // 성인 콘텐츠 여부 (true : 성인콘텐츠, false : 일반콘텐츠)
    private String posterUrl; // 썸네일 url
    private String status; // 다운로드 상태 (ADDED : 다운로드 대기중, PAUSED : 다운로드 일시중지, PROGRESS : 다운로드 중, COMPLETED : 다운로드 완료, DELETED : 다운로드 삭제됨)
    private int progress; // 다운로드 진행률
    private boolean checkState; // + 체크박스 상태값 체크 (true : 체크된 상태, false : 체크해제 상태)

    public DownloadDto(String contId, String ty1Code, String contNm, boolean isAdultCont, String posterUrl) {
        this.contId = contId;
        this.ty1Code = ty1Code;
        this.contNm = contNm;
        this.isAdultCont = isAdultCont;
        this.posterUrl = posterUrl;
        this.status = DownloadManager.ADDED;
        this.progress = 0;
        checkState = false;
    }
}
```

2. 다운로드 목록 화면 (DownloadFragment Class)



*옵저버 (DownloadFragment Class)

DownloadManager Class에서 실행중인 무한루프의 Thread가 0.5초마다 download() 메소드를 호출하면서 DownloadDto Progress 값의 변수를 계속해서 5씩 증가시키는데 (최대 100까지) 이 때 실시간으로 옵저버가 값이 바뀌고 있는 Dto 객체를 감지하여 뷰에 갱신을 한다 -> 프로그레스바의 분홍색게이지, 버튼아이콘

```
final Observer<DownloadDto> downloadDtoObserver = downloadingDto -> { // 옵저버
    for (DownloadDto adapterDto : adapter.getAdapterList()) {
        if (adapterDto.getContId().equals(downloadingDto.getContId())) {
            adapterDto.setStatus(downloadingDto.getStatus());
            adapterDto.setProgress(downloadingDto.getProgress());
            adapter.notifyItemChanged(adapter.getAdapterList().indexOf(adapterDto));
        }
    }
    binding.setVisibleData(!adapter.getAdapterList().isEmpty());
};

viewModel.getCurrentDownloadDtoState().observe(getViewLifecycleOwner(), downloadDtoObserver); // 옵저버 변화감지
```

*ViewModel 클래스를 상속받은 DownloadViewModel Class의 LiveData 코드 일부분

```
public class DownloadViewModel extends ViewModel {
    private MutableLiveData<DownloadDto> currentDownloadDtoState = new MutableLiveData<>();

    public LiveData<DownloadDto> getCurrentDownloadDtoState() {
        return currentDownloadDtoState;
    }

    private DownloadStatusListener downloadStatusListener = new DownloadStatusListener() {
        @Override
        public void onAdded(DownloadDto downloadDto) {
            Log.d( tag: "===onAdded | ", msg: downloadDto.getContId() + " | " + downloadDto.getStatus() + " | " + downloadDto.getProgress());
            currentDownloadDtoState.postValue(downloadDto);
        }

        @Override
        public void onPaused(DownloadDto downloadDto) {
            Log.d( tag: "===onPaused | ", msg: downloadDto.getContId() + " | " + downloadDto.getStatus() + " | " + downloadDto.getProgress());
            currentDownloadDtoState.postValue(downloadDto);
        }

        @Override
        public void onProgress(DownloadDto downloadDto) {
            Log.d( tag: "===onProgress | ", msg: downloadDto.getContId() + " | " + downloadDto.getStatus() + " | " + downloadDto.getProgress());
            currentDownloadDtoState.postValue(downloadDto);
        }
    }
}
```

*DownloadManager Class

```
public void activeDownloadThread() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                while (true) { // 무한루프
                    Log.d( tag: "DownloadThread", msg: "isActive");

                    if (recentDownload != null) { // 현재 다운로드하고 있는 객체가 있으면 다운로드 메소드 계속 호출해서 실행 -> while(true) 무한루프
                        download(recentDownload);
                    } else { // 다운로드를 다 끝마친 뒤 recentDownload 객체가 null 이 될 경우 else 문에서 다음 "DownloadDto" 객체를 download 메소드를 호출하여 실행시키고 없으면 for 문을 빠져나온
                        for (DownloadDto downloadDto : downloadQueue) { // 다운로드 대기열 리스트에 있는 "DownloadDto" 객체중
                            if (TextUtils.equals(ADDED, downloadDto.getStatus()) // status 변수가 ADDED 이거나 PROGRESS 상태인 "DownloadDto" 객체를 다시 다운로드 메소드 호출해서 실행
                                || TextUtils.equals(PROGRESS, downloadDto.getStatus())) {
                                download(downloadDto);
                                break;
                            }
                        }
                    }
                }
            } catch (Exception e) {}
        }
    }).start();
}

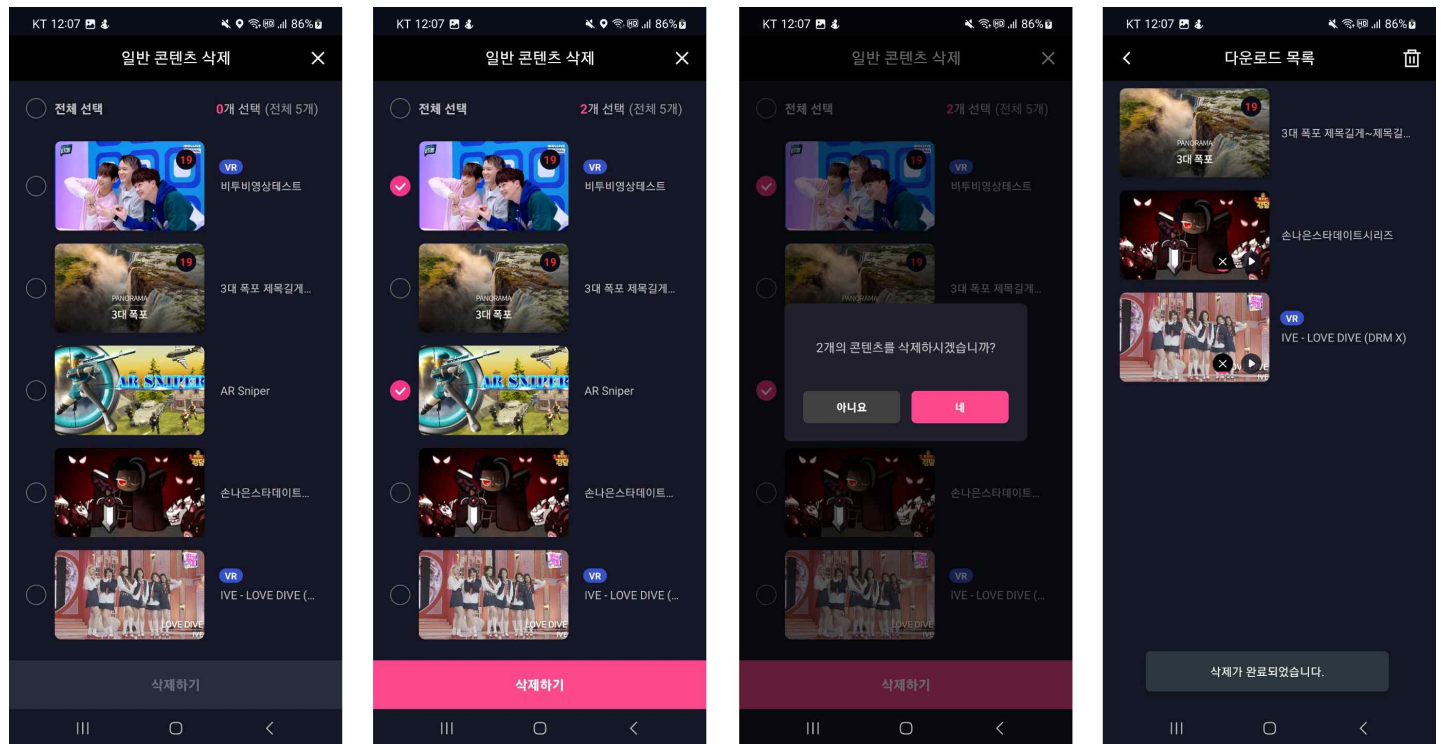
private void download(DownloadDto downloadDto) {
    int progress = downloadDto.getProgress();

    if (progress < 100) {
        downloadDto.setProgress(progress + 5);
        downloadDto.setStatus(PROGRESS);
        recentDownload = downloadDto;
        onCallBack(downloadDto, PROGRESS);
    } else {
        downloadDto.setStatus(COMPLETED);
        recentDownload = null;
        onCallBack(downloadDto, COMPLETED);
    }

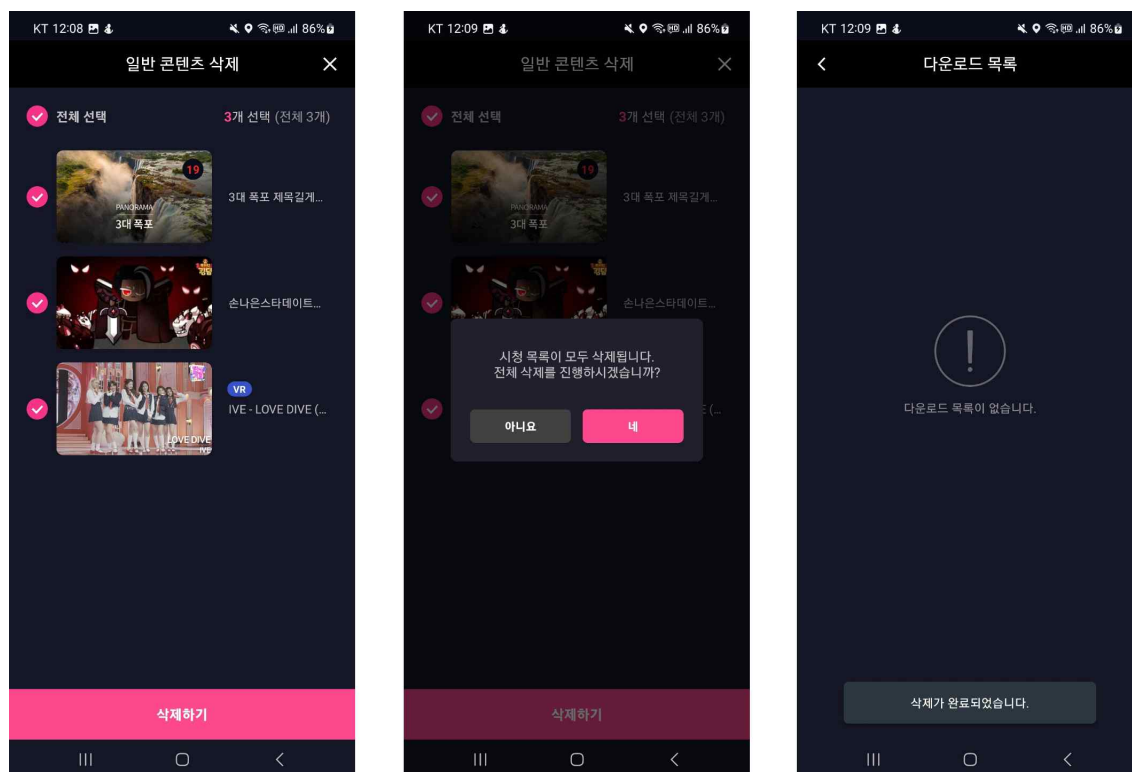
    Log.d( tag: "===Thread | ", downloadDto.toString());
}
```

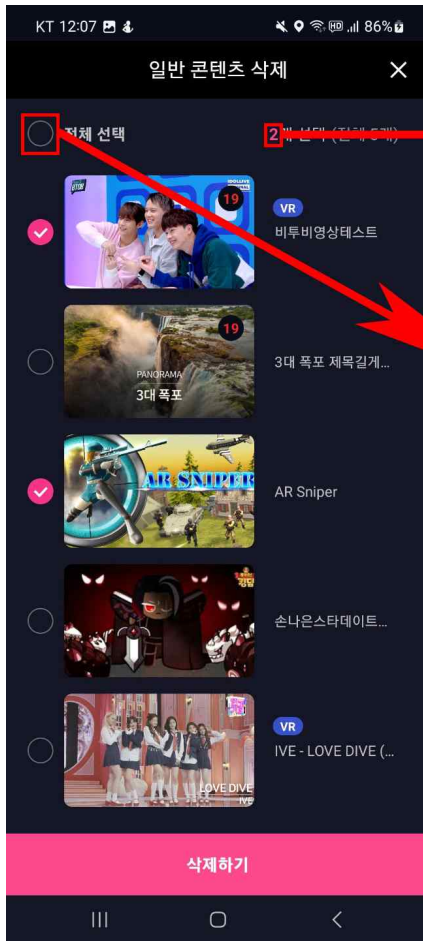
3. 삭제 목록 화면 (DeleteFragment Class)

*개별 삭제 시



*전체 삭제 시





```
adapter.setOnItemClickListener((v, isSelect) -> { // !체크박스 개별선택 버튼 (클릭)
    if (isSelect) {
        count++;
        if (count == adapter.getAdapterList().size()) { binding.deleteCheckAllBtn.setChecked(true); }
    } else {
        count--;
        binding.deleteCheckAllBtn.setChecked(false);
    }
    binding.contentCountText.setText(String.valueOf(count)); // count 변수 값 setText
    binding.itemDeleteBtn.setEnabled(count > 0); // 삭제하기 버튼 활성화/비활성화
});
```

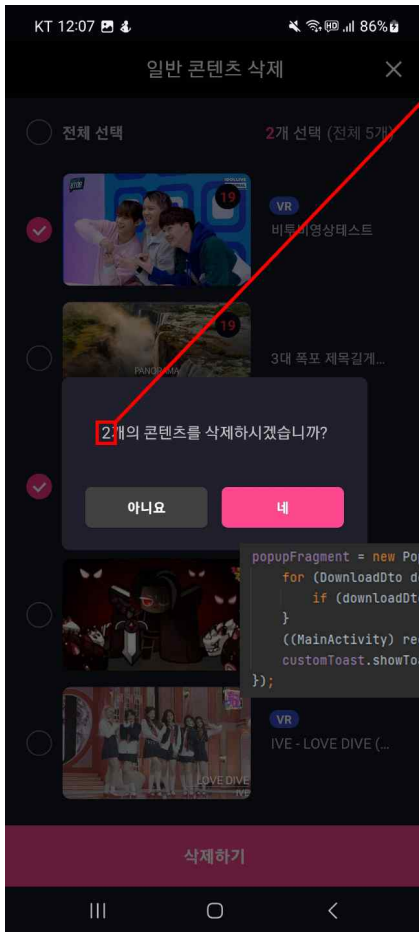
RecyclerView 아이템의 체크박스의 클릭이벤트가 발생할때마다 int 타입의 count 변수의 값이 증가하거나 감소하며 TEXT와 삭제하기 버튼을 활성화/비활성화 시킨다

```
binding.deleteCheckAllBtn.setOnClickListener(view -> { // @체크박스 전체선택 버튼 //
    if (binding.deleteCheckAllBtn.isChecked()) {
        checkAll( setCheckStateAll: true);
        setButtonAndCount(adapter.getAdapterList().size());
    } else {
        checkAll( setCheckStateAll: false);
        setButtonAndCount(0);
    }
});
```

*해당 클래스에서 만든 커스텀 메소드

```
public void checkAll(boolean setCheckStateAll) { // "DownloadDto" 객체의 CheckState 변수를 바꿈
    for (DownloadDto downloadDto : adapter.getAdapterList()) {
        if (setCheckStateAll) {
            if (!downloadDto.getCheckState()) { downloadDto.setCheckState(true); }
        } else {
            if (downloadDto.getCheckState()) { downloadDto.setCheckState(false); }
        }
    }
    adapter.notifyDataSetChanged();
}
```

```
public void setButtonAndCount(int contentCount) {
    count = contentCount;
    binding.contentCountText.setText(String.valueOf(count));
    binding.itemDeleteBtn.setEnabled(count > 0);
}
```



```
binding.itemDeleteBtn.setOnClickListener(view -> { // @삭제하기 버튼 //
    if (count == adapter.getAdapterList().size()) { showDialog("시청 목록이 모두 삭제됩니다.\n전체 삭제를 진행하시겠습니까?"); }
    else { showDialog(count + "개의 콘텐츠를 삭제하시겠습니까?"); }
});
```

*showDialog()는 해당 클래스(DeleteFragment)에서 만든 커스텀 메소드

```
public void showDialog(String message) {
    popupFragment.setMessage(message);
    popupFragment.setCancelable(false);
    popupFragment.show(requireActivity().getSupportFragmentManager(), tag: "");
}
```

PopupFragment는 DialogFragment를 상속받은 객체로서 setMessage()메소드를 통해 파라미터로 넘겨받은 String값으로 화면에 띄울 메시지를 출력한다

```
public interface OnDialogClickListener {
    void onClickConfirmButton();
}

public class PopupFragment extends DialogFragment {
    private FragmentPopupBinding binding;
    private final OnDialogClickListener listener;
    private String message;

    public PopupFragment(OnDialogClickListener listener) { this.listener = listener; } // 생성자

    public void setMessage(String message) { this.message = message; }
```

```
popupFragment = new PopupFragment() -> { // PopupFragment (DialogFragment를 상속받은) 객체 생성 -> !삭제팝업창에서 확인버튼을 눌렀을 때 실행 (OnDialogClickListener)
    for (DownloadDto downloadDto : adapter.getAdapterList()) {
        if (downloadDto.getCheckState()) { viewModel.removeDownload(downloadDto); }
    }
    ((MainActivity) requireActivity()).moveTo(new DownloadFragment()); // 다운로드목록으로 이동
    customToast.showToast("삭제가 완료되었습니다.");
};
```

PopupFragment 객체의 '네' 버튼 클릭시 ViewModel에서 정의한 removeDownload()메소드를 호출하여 DownloadDto 객체의 Boolean 타입의 CheckState 변수가 true인 Dto 객체를 삭제한다

```
binding.yesBtn.setOnClickListener(view1 -> { // @네 버튼 //
    listener.onClickConfirmButton();
    dismiss();
});

binding.noBtn.setOnClickListener(view12 -> dismiss()); // @아니오 버튼 //
```

***RecyclerView Adapter

RecyclerView의 getItemViewType()메소드를 오버라이드하여 2개의 RecyclerView ViewHolder를 사용
(이너 클래스 : 다운로드 목록 뷰홀더, 삭제 목록 뷰홀더)

```
public class U DiveRecyclerViewAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {
    private int itemType;
    public static final int VIEW_TYPE_WATCH = 0, VIEW_TYPE_DELETE = 1;
    private ArrayList<DownloadDto> list;
    private OnItemClickListener itemClickListener;
    private OnItemOrderListener itemOrderListener;

    @Override
    public int getItemViewType(int position) { return itemType; }
```

*아래의 setItemViewType은 커스텀 메소드

```
public void setItemViewType(int itemType) { this.itemType = itemType; } // 뷰 타입 변경
```

- 다운로드 목록 뷰홀더

```
private static class DownloadListViewHolder extends RecyclerView.ViewHolder { // 다운로드 목록 클래스
    itemViewDownloadListBinding binding;
    OnItemOrderListener orderListener;
    DownloadDto dto;
    ImageView imageView;
    ClipDrawable clipDrawable;

    public DownloadListViewHolder(@NonNull itemViewDownloadListBinding binding, OnItemOrderListener orderListener) { // 다운로드 목록 생성자
        super(binding.getRoot());
        this.binding = binding;
        this.orderListener = orderListener;
        imageView = itemView.findViewById(R.id.progress_circular);
        clipDrawable = (ClipDrawable) imageView.getBackground();

        binding.startAndPauseBtn.setOnClickListener(view -> { // 재생/일시정지 버튼
            if (binding.startAndPauseBtn.isChecked() && !dto.getStatus().equals(DownloadManager.PROGRESS)) {
                binding.startAndPauseBtn.setChecked(true);
                orderListener.onItemBeginOrder(itemView, dto, order: "start");
            } else {
                binding.startAndPauseBtn.setChecked(false);
                orderListener.onItemBeginOrder(itemView, dto, order: "pause");
            }
        });

        binding.downloadDeleteBtn.setOnClickListener(view -> orderListener.onItemBeginOrder(itemView, dto, order: "remove")); // 다운로드 삭제 버튼
    }

    public void bind(DownloadDto item) {
        dto = item;
        binding.setItem(item); // 타이틀, 성인콘텐츠, 콘텐츠타입, 썸네일, 아이템버튼, 프로그레스바를
        clipDrawable.setLevel(item.getProgress() * 100); // 프로그레스
    }
}
```

Windows 정품 인증

- 삭제 목록 뷰홀더

```
private static class DeleteListViewHolder extends RecyclerView.ViewHolder { // 삭제목록 클래스
    ItemViewDeleteListBinding binding;
    OnItemClickListener clickListener;
    DownloadDto dto;

    public DeleteListViewHolder(@NonNull ItemViewDeleteListBinding binding, OnItemClickListener clickListener) { // 삭제목록 생성자
        super(binding.getRoot());
        this.binding = binding;
        this.clickListener = clickListener;

        binding.getRoot().setOnClickListener(view -> {
            dto.setCheckState(!binding.checkbox.isChecked());
            clickListener.onItemClick(itemView, !binding.checkbox.isChecked());
            binding.checkbox.setChecked(!binding.checkbox.isChecked());
        });

        binding.checkbox.setOnClickListener(view -> {
            dto.setCheckState(binding.checkbox.isChecked());
            clickListener.onItemClick(itemView, binding.checkbox.isChecked());
        });
    }

    public void bind(DownloadDto item) {
        dto = item;
        binding.setItem(item); // 타이틀, 성인콘텐츠, 콘텐츠타입, 썸네일, 아이템 체크박스
    }
}
```

*RecyclerView 추상메소드

onCreateViewHolder()시 클래스에서 정의한 int타입의 itemType 변수의 값에 따라서 이너클래스인 DownloadListViewHolder 클래스 객체나 DeleteListViewHolder 클래스 객체를 생성하고 onBindViewHolder()시 JAVA의 instanceof로 객체의 부모를 판별해서 해당 부모클래스의 bind()메소드를 호출, 아이템을 바인딩해서 화면에 보여준다

```
@NonNull
@Override
public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    if (itemViewType == VIEW_TYPE_WATCH) {
        ItemViewDownloadListBinding downloadListBinding = ItemViewDownloadListBinding.inflate(inflater, parent, attachToRoot false);
        return new DownloadListViewHolder(downloadListBinding, itemOrderListener);
    } else {
        ItemViewDeleteListBinding deleteListBinding = ItemViewDeleteListBinding.inflate(inflater, parent, attachToRoot false);
        return new DeleteListViewHolder(deleteListBinding, itemClickListener);
    }
}

@Override
public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {
    if (holder instanceof DownloadListViewHolder) { ((DownloadListViewHolder) holder).bind(list.get(position)); }
    else ((DeleteListViewHolder) holder).bind(list.get(position));
}

@Override
public int getItemCount() { return list.size(); }
```


*DataBinding & BindingAdapter

BindingAdapters 클래스의 바인딩어댑터 코드 일부

@BindingAdapter("XML파일에서 사용할 기능명") 어노테이션과 public static으로 선언해 XML에서 해당 클래스에서 정의한 메소드를 사용할 수 있다

```
public class BindingAdapters {
    // region DownloadLayoutView
    @BindingAdapter("recyclerViewVisible")
    public static void setRecyclerView(RecyclerView recyclerView, boolean existData) {
        int visibility = existData ? View.VISIBLE : View.INVISIBLE;
        recyclerView.setVisibility(visibility);
    }

    @BindingAdapter("dataDeleteButtonVisible")
    public static void setDeleteButtonView(Button button, boolean existData) {
        int visibility = existData ? View.VISIBLE : View.INVISIBLE;
        button.setVisibility(visibility);
    }

    @BindingAdapter("imageVisible")
    public static void setImageView(ImageView imageView, boolean existData) {
        int visibility = existData ? View.INVISIBLE : View.VISIBLE;
        imageView.setVisibility(visibility);
    }

    @BindingAdapter("textVisible")
    public static void setTextView(TextView textView, boolean existData) {
        int visibility = existData ? View.INVISIBLE : View.VISIBLE;
        textView.setVisibility(visibility);
    }
    // endregion
}
```

- XML파일 일부 코드

```
<?xml version="1.0" encoding="utf-8"?>
<layout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>
        <variable
            name="visibleData"
            type="java.lang.Boolean" />
    </data>
```

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:orientation="vertical"
    tools:listitem="@layout/item_view_download_list"
    app:layout_constraintTop_toBottomOf="@id/title"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    recyclerViewVisible="@{visibleData}" />
```