There are better options than using one array for names and another array for resources.

acarlstein.com/

Posted by Alejandro G. Carlstein Ramos Mejia on August 20, 2018 August 20, 2018 About Programming / Android / Java

I encounter a piece of code which bugs me:

```
int[] images = {
  R.drawable.ic_image_1, R.drawable.ic_image_2, R.drawable.ic_image_3,
R.drawable.ic_image_4, R.drawable.ic_image_5
  R.drawable.ic_image_6, R.drawable.ic_image_7, R.drawable.ic_image_8,
R.drawable.ic_image_9, R.drawable.ic_image_10
};

String[] imagesNames = {
 "Image One", "Image Two", "Image Three", "Image Four", "Image Five",
 "Image Six", "Image Seven", "Image Eight", "Image Nine", "Image Ten"
};
```

Later on, these two array are pass to all kind of method and constructors together for example:

```
    @Override
    public void onBindViewHolder(MasonryView holder, int position) {
        holder.textView.setText(imagesNames[position]);
        holder.imageView.setImageResource(images[position]);
    }
```

While this is correct, light on the memory, and use fewer CPU resources (less like of code run, smaller Big O), it welcomes all short of possible bugs, for example, lets say you add an image and forgot to add a name. You get mismatch arrays. Your code breaks. This issue is common if you are building these arrays dynamically; therefore, why not keeping the information together?

There are different ways to handle this

One way would be to create a class which holds these two items (image and image name). Then, you can create an array of objects.

```
private class Resource {
 public int image;
 public String name;
 Resource(String name, int image){
  this.image = image;
  this.name = name;
 }
}

Resource[] resources = {
 new Resources("Image One", R.drawable.ic_image_1),
 new Resources("Image Two", R.drawable.ic_image_2),
 new Resources("Image Three", R.drawable.ic_image_3),
 ...
 new Resources("Image Ten", R.drawable.ic_image_10),
}
```

Then, you can iterate them:

```
    @Override
    public void onBindViewHolder(MasonryView holder, int position) {
        holder.textView.setText(resources[position].name);
     holder.imageView.setImageResource(resources[position].image);
    }
```

Now, this is cleaner than before.

However, lets say you don't wish to use a (public or private) class, then what to do? Well, you could use our old friend LinkedHashMap, which works like a HashMap; however, it keeps the order in which the elements were inserted into it.

```
LinkedHashMap = new LinkedHashMap<String, Integer>();
resources.put("One", R.drawable.ic_image_1);
resources.put("Two", R.drawable.ic_image_2);
...
resources.put("Ten", R.drawable.ic_image_10);
```

Now, here there is a problem. While you can iterate each of these items and get them based on the key (name), you cannot access them with an index. Do not despair! There is a solution around it by using the entrySet() method which is offer with the LinkedHashMap. The entrySet() method returns a set view of the mappings contained in the map. Lets see how it can help us:

```
resourcesIndexed = new ArrayList<Map.Entry<String, Integer>>(resources.entrySet());

    @Override
    public void onBindViewHolder(MasonryView holder, int position) {
        holder.textView.setText(resourcesIndexed.get(position).getKey());
        holder.imageView.setImageResource(resourcesIndexed.get(position).getValue());
    }
```

As you can see, without creating a container, such as a class, we can use an index to obtain the information and keep track of each pair.