# Quiz 6 Review:

## 9.1 Lists, Loop Idioms, Generics, and the Null Keyword

### 9.1.1

* Lists are resizable, ordered collections of data. They solve problems that fixed-size structures (arrays) make award.
  + A good example is a book.
  + List<Book> contents = new ArrayList<>(); //grows and shrinks as needed
    - Random ccess with get(i)
    - Variable Length - no need to predict capacity.
    - Rich API features.

### 9.1.2: Loop Idioms table

| Idiom | Purpose | Skeleton |
| --- | --- | --- |
| Counter-Controlled | Repeat n times i iterate by index | for (int i=0; i<n; i++) |
| For-Each | iterate over values in a collection | for (Book b ” contents)“) |
| Early Exit | stop as soon as an answer found | return … or break; |
| Accumulate | Build up a result (Sum, list, count) | initialize Accumulator -> update inside loop |
| ——————– | ————————————– | ———————————————- |

### 9.1.5: Generics Revisited

Sometimes in java there are cases when you;ll need to create methods without knowing what type of data youll be working with or, more properly, so theyll work with any type of data. This shit is retarded and should not exist and is an affrond to human intelligence. For instances where you desire to be a sub-human piece of trash, you can use a Generic Type Parameter to represent the type of data. Declaring classes that use this require using a new syntax to refer to the class name (wow!) which uses angle brackets (<…>) containing one or more variable (sep by a comma) to refer to unspecified type names. For example, you would use or <Key, Value> to refer to unspecified type names. This eliminates explicit casts, catches type errors at compile time (cool i guess) This is how it is used and its syntax:

public class Box <Content>  
{  
 private Content value;  
  
 public Box(Content newValue)  
 {  
 this.value = newValue;  
 }  
  
 public Content getValue()  
 {  
 return this.value;  
 }  
  
 public void setValue(Content newValue)  
 {  
 this.value = newValue;  
 }  
}

After that you can make a new Box object:

Box<String> box1 = new Box<String>("surprise");  
Box<Integer> box2 = new Box<Integer>(42);

### 9.1.8: Null Keyword

When you declare an object variable, you are storing a reference to an object. In java, the keyword null means no object. you can delcare and initialize object variables this way. Any derefencing (x.f, x.m()) when x == null is a NullPointerException (NPE). - You’e dereferencing a null pointer.

## 10.01 - Arrays:

Lists are used when there is a need to story much, but an unknown amount of data in a single object, but there is also Arrays. Arrays are more primitive, and are one of the principle building blocks of data structures in many programming languages. An array is a named collection of contiguous storage locations-storage locations that are next to each other that contain data items of the same type. The index is the list effectively.

### 10.1.1: Basics

| Feature | Array | ArrayList |
| --- | --- | --- |
| Size | Fixed | Dynamic |
| Syntax | int[] a = new int[5]; | List a = new ArrayList<>(); |
| Length | a.length (field) | a.size() (Method) |
| Default Values | 0, 0.0, false, null | N/A |
| Methods | none (only fields) | Shit ton |
| —————- | ———————– | ————————————– |

#### Creating an Array:

can be made with any data type (primitve).

int[] counts = new int[4]; // [0,0,0,0]  
String[] words = {"one", "Two"} // literally that

#### Indexing and ASsignment:

counts[0] = 7 int x = counts[i+1];

### 10.1.2: Looping over arrays

for (int i=0; i<values.length; i++) {  
 values[i] = 2\*i  
}  
  
for (String w : words) {  
 System.out.println(w);  
}

### 10.1.3 General:

#### Alias

* Point to the same array.

int[] a = new int[3];  
int[] b = a;

### 10.1.4 Itializing large arrays programmatically:

int[] seq = new int[100];  
for (int i=0; i<seq.length; i++) seq[i] = i;

### 10.1.5 Modulo (%) and Integer Division

int feet = inches / 12;  
int in = inches % 12;  
if (x % 5 == 0)

# 11.01. Multi-demensional Arrays

## 11.1.1: 2-D Arays (Arays of Arrays)

double[][] rain = new doub;e[13][32];  
rain[1][5] = 1.15;

* First index = row (Month), second = column (day)
* Lengths: rain.length == 13; rain[month].length == 32.

## 11.1.2: 3D and higher arrays:

double[][][] rain10y = new double[10][13][32]; //year, month, day

## 11.1.3: Jagged (Ragged) Arrays

int[][] jag = {  
 {1,2,3},  
 {4,5},  
 {6,7,8,9}  
}

## 11.1.4: Multi-dimensional lists

A List<List> grid = new ArrayList<>(); gives similar flexibility but with dynamic sizing. Access via grid.get(r).get(c)