

# Integer and Double

Classes you can use!

# ArrayList Goofiness

- You can only put instances of classes as the types for `ArrayList<E>`

```
ArrayList<String> list = new ArrayList<String>();  
ArrayList<Bug> list = new ArrayList<Bug>();
```

- You can't use primitives

```
ArrayList<int> list = new ArrayList<int>();  
ArrayList<double> list = new ArrayList<double>();
```

# ArrayList Goofiness

- You can only put instances of classes as the types for ArrayList<E>

```
ArrayList<String> list = new ArrayList<String>();  
ArrayList<Bug> list = new ArrayList<Bug>();
```

- You can't use primitives

```
ArrayList<int> list = new ArrayList<int>();  
ArrayList<double> list = new ArrayList<double>();
```

# Solution

- Make an Integer and Double class!

```
ArrayList<Integer> list = new ArrayList<Integer>();  
ArrayList<Double> list = new ArrayList<Double>();
```

# Integer class basics

```
public class Integer {  
    private int x;  
  
    public Integer(int val) {  
        x = val;  
    }  
  
    public int intValue() {  
        return x;  
    }  
}
```

# Integer's in Action

```
public static void main(String[] args) {  
    ArrayList<Integer> list = new ArrayList<Integer>();  
    Integer val = new Integer(3);  
  
    list.add(val);  
    list.add(new Integer(5));  
  
    Integer num = list.get(1);  
  
    System.out.println(num); //5  
}
```

# Autoboxing

- You can treat Integer's as int's!

```
public static void main(String[] args) {  
    Integer x = new Integer(3);  
    Integer y = new Integer(4);  
  
    int z = x.intValue() + y.intValue(); //7  
    int a = x + y; //7 too!  
  
    Integer b = 4; //also works!  
    int c = x; //works too!  
}
```

# For-Each Loop

- Special For Loop
- Only useful if you want to visit every element in an `ArrayList` (or array) one after the other
- Not useful if you need to modify an `ArrayList/array`

# For-Each

```
ArrayList<Double> list = ...;
```

```
//v1
```

```
double sum = 0;  
for(int i = 0; i < list.size(); i++)  
    sum += list.get(i);
```

```
//v2
```

```
double sum = 0;  
for(double x: list)  
    sum += x;
```

# Good/Bad

- Advantages
  - Shorter/Easier to read code
  - Never out of bounds!
- Disadvantages
  - Can't modify list while looping
  - Don't know indexes

# For-Each Examples

- `ArrayList<Integer> nums = ... //{4, 1, 3, 2, 0}`

```
for(int x: nums)
    System.out.println(nums.get(x));
```

# For-Each Examples

- `ArrayList<Integer> nums = ... //{4, 1, 3, 2, 0}`

```
for(int x: nums)
    System.out.println(nums.get(x));
```

`x=4..... 0`

# For-Each Examples

- `ArrayList<Integer> nums = ... //{4, 1, 3, 2, 0}`

```
for(int x: nums)
    System.out.println(nums.get(x));
```

```
x=4..... 0
```

```
x=1..... 1
```

# For-Each Examples

- `ArrayList<Integer> nums = ... //{4, 1, 3, 2, 0}`

```
for(int x: nums)
    System.out.println(nums.get(x));
```

`x=4..... 0`

`x=1..... 1`

`x=3..... 2`

# For-Each Examples

- `ArrayList<Integer> nums = ... //{4, 1, 3, 2, 0}`

```
for(int x: nums)
    System.out.println(nums.get(x));
```

`x=4..... 0`

`x=1..... 1`

`x=3..... 2`

`x=2..... 3`

# For-Each Examples

- `ArrayList<Integer> nums = ... //{4, 1, 3, 2, 0}`

```
for(int x: nums)
    System.out.println(nums.get(x));
```

`x=4..... 0`

`x=1..... 1`

`x=3..... 2`

`x=2..... 3`

`x=0..... 4`

# For-Each Can't Remove

# For-Each Can't Remove

```
ArrayList<Integer> values = ...
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...  
for(int x: values)
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...  
for(int x: values)  
    if(x == 3)
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...  
for(int x: values)  
    if(x == 3)  
        values.remove(x); //Concurrent Modification Exception
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...  
for(int x: values)  
    if(x == 3)  
        values.remove(x); //Concurrent Modification Exception
```

```
ArrayList<Integer> nums = ...
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...  
for(int x: values)  
    if(x == 3)  
        values.remove(x); //Concurrent Modification Exception
```

```
ArrayList<Integer> nums = ...  
for(int i = nums.size()-1; i>=0; i--)
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...  
for(int x: values)  
    if(x == 3)  
        values.remove(x); //Concurrent Modification Exception
```

```
ArrayList<Integer> nums = ...  
for(int i = nums.size()-1; i>=0; i--)  
{
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...  
for(int x: values)  
    if(x == 3)  
        values.remove(x); //Concurrent Modification Exception
```

```
ArrayList<Integer> nums = ...  
for(int i = nums.size()-1; i>=0; i--)  
{  
    int value = nums.get(i);
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...  
for(int x: values)  
    if(x == 3)  
        values.remove(x); //Concurrent Modification Exception
```

```
ArrayList<Integer> nums = ...  
for(int i = nums.size()-1; i>=0; i--)  
{  
    int value = nums.get(i);  
    if(value == 3)
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...  
for(int x: values)  
    if(x == 3)  
        values.remove(x); //Concurrent Modification Exception
```

```
ArrayList<Integer> nums = ...  
for(int i = nums.size()-1; i>=0; i--)  
{  
    int value = nums.get(i);  
    if(value == 3)  
        nums.remove(i); //not nums.remove(value)!}
```

# For-Each Can't Remove

```
ArrayList<Integer> values = ...
for(int x: values)
    if(x == 3)
        values.remove(x); //Concurrent Modification Exception
```

```
ArrayList<Integer> nums = ...
for(int i = nums.size()-1; i>=0; i--)
{
    int value = nums.get(i);
    if(value == 3)
        nums.remove(i); //not nums.remove(value)!
}
```