Generics

Generics allows you to write code that can be **reused** with different data types. This allows for versatile code that works for any data type we put into it. Because of this, the compiler will be more strict on data types and we would find more bugs. This also means we **do not have to cast**.

Generics Example (Array Lists)

For example, lets say that we wanted to code a backpack to put things in. The **Generics version** with **type safety** would be:

```
1 ArrayList<Integer> backpack = new ArrayList<integer>();
2 backpack.add(new Integer(4));
```

Here, we have an `ArrayList` that only accepts `Integer` type data. If we put in a `String`, it would error.

When we dont do generics, such as:

```
1 ArrayList backpack = new ArrayList();
2 backpack.add(new Integer(4));
3 backpack.add("hi");
```

The first example would cast every element inside the array into type `Integer`. However, it does not take `String` type values. This is using a typed generic. In the second one, since there is not a declared type, it will accept any type of data but store it as an `Object`. To get an array out of an `Object`, we would need to first **Change the object 4 into an integer 4**:

```
Integer myInt = (Integer) backpack.get(0); //Works for example 2
```

Whereas in the firs example, **We already declared the type in the beginning**, so we would not have to **cast**, or change 4 into an integer object:

```
Integer myInt = backpack.get(0); //Works for example 1
```

Why do this?

When we created the generic `<Integer>` to `ArrayList`, we are only allowing intergers to be put in that list. If we try to put a `String` in there, it would error. For example, if we tried to run line 3 of example 2 (the non-generic list code) in example 1 (the generic `ArrayList`), we would get a **Compile error**, which means it would not compile. This is better than a **RunTime error**, meaning it errors when we are running the program. Compile errors are alot easier to fix than runtime errors because they can be found by the compiler, which then tells you which line is causing the problem, and fixed. In runtime errors, a program crashes because it does something that causes it to crash. Java will not tell you wich line you crashed on (Unless you use a debugger), leaving you to hair pick your code and look for the bug.

Writing Generic Classes

We can write classes that take generic types and then work with that type throughout the class. This will save us time from restructuring the whole class to meet the user's needs:

```
1 public class Backpack<E> {
2   ...
3 }
```

Now the backpack will take any type (put into something like a variable into `<E>`) and be able to perform. Now, we're going to create an `ArrayList` to contain our backpack items.

```
1 public class Backpack <E> {
2
```

```
  private ArrayList<E> container;

  //Constructor
  public Backpack() {
    container = new ArrayList<E>();
  } //End constructor
}
```