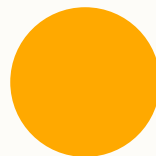




Generics and Types

Reese Hatfield



0



What is a type?

- Data-types
 - Primitives (int, double, etc)
 - Wrapper (Integer, Double, etc)
- CustomArrayList
 - How would you handle supporting different types?





What is a type?

- Overloaded methods?
 - Very clunky
 - Significant code duplication
- Object class?
 - Lose type information
 - Would need to cast/check everywhere





What is a type?

- What if we could define the datatype itself as a parameter?
- What if we could do something like:
- `void doThing(parameters, return Type)`
- Reuse the body of the code





Generics

- This is what generics let us do
- Usually defined with `<>`
- You've used them already
 - `ArrayList<Type>`
 - `Queue<Type>`
 - etc.





Generics

- Change a datatype
 - *At compile time*
- Compile time change ensures that we can't change the type later
 - Object class would not promise this



Essence of a Type

```
String s = new String("content")
```

Type

Memory Content



Essence of a Type

```
String s = new String("content")
```

Type

Memory Content

```
ArrayList<String> c = new ArrayList<>();
```




Essence of a Type

```
String s = new String("content")
```

Type

Memory Content

```
ArrayList<String> c = new ArrayList<>();
```

Type

Holder
Type

Memory Content



Generics

- Usually two places where you can define generics
 - Generic methods (rare)
 - Generic classes (common)
- `public <T> void print(T item){...}`
- `public class Box<T> {...}`





Generics

```
public class Box<T>
```

- "T" becomes the type parameter
- T → whatever it's defined as

```
Box<Integer> = new Box<>(5)
```

- It becomes an integer in this case





Generics

- We can have as many of these as we want!

```
Box<Integer> = new Box<>(5)
```

```
Box<String> = new Box<>("abc")
```





Generics

- "T" is common for "Type"
 - Can use whatever makes sense
- You'll commonly see:
 - T (type)
 - K (key)
 - V (value)
 - E (element)





Generics

- Can include more than one parameter type
- `public class Thing<T1, T2>`
- Let's see that!





Pair

- I want a class like Box, but it can hold two types
- Pair?
- I would like my pairs to only have the first be a number type





Pair

- How can I sort something if I don't know the type ahead of time?
- I only know it once it has been created
- But Java itself *is* aware of the type
- Want to not compile if type is invalid





Pair

- Use the *extends* keyword
- Limit the type to only types that are a subtype of what we want
- Bounded Type Parameters
- `<T extends Number>`
- `<T extends String>`





Pair

- Extends even when using an interface
 - `<T extends Comparable<T>>`
- Comparable itself is an generics
- Let's use this to make our Pair class sortable





Pair

- Let's use this for something more useful!
- Let's write our own ArrayList class
- CustomArrayList<T>

