Lecture 19: Issues in Copying

Last time:
1. Intro to arrays
2. Copying arrays and making arrays bigger
3. Array lengths and out-of-bounds indexing
4. Passing arrays and array elements to a function

This lecture set:
1. Privacy Leaks
2. Different levels of copy
Privacy Leaks (continued)

```java
public class MutableThing {
    ...  
    public void mutateMe() {...};
}

public class Foo {
    private MutableThing q  
        = new MutableThing();
    ...
    public MutableThing getQ(){
        return q;
    }
}
```

- Consider following code
  ```java
  Foo f = new Foo();
  MutableThing m = f.getQ();
  m.mutateMe();
  ```
- After this executes, what happens?
- This phenomenon is called a privacy leak
  - Private instance variables can be modified outside class
  - Behavior is due to aliasing
Fixing Privacy Leaks

- Return copies of objects referenced by instance variables
- To fix `getQ` method in `Foo`:
  ```java
  MutableThing getQ(){
      return new MutableThing(q);
  }
  ```
  - This returns a copy of `q`
- Changes made to this copy will not affect original
Reference Copying

Person[] d = {
    new Person("SGH", ...),
    new Person("Shakira", ...)
};

Person[] e = d;

Stack

Heap

SGH

Shakira
Shallow Copying

Person[] d = {
    new Person(“SGH”, …),
    new Person(“Shakira”, …)
};

Person[] e = new Person[d.length];
for (int i=0; i < d.length, i++){
    e[i] = d[i];
}
Deep Copying

```java
Person[] d = {
    new Person("SGH", ...),
    new Person("Shakira", ...)
};

Person[] e = new Person[d.length];
for (int i=0; i<d.length; i++) {
    e[i] = new Person(d[i]);
}
```
Three Ways of Copying

CDCollector contains an array of CD’s;
ReCDCollector contains an array of rewritableCD’s;

- **Reference copy**

```java
public ReCD[] getCDsReferenceCopy() {
    return myFavorites;
}
```

- **Shallow copy**

```java
public ReCD[] getCDsShallowCopy() {
    ReCD[] copy = new ReCD[myFavorites.length];
    for (int i = 0; i < copy.length; i++)
        copy[i] = myFavorites[i];
    return copy;
}
```

- **Deep copy**

```java
public ReCD[] getCDsDeepCopy() {
    ReCD[] copy = new ReCD[myFavorites.length];
    for (int i = 0; i < copy.length; i++)
        copy[i] = new ReCD(myFavorites[i]);
    return copy;
}
```
When To Use What Kind of Copying?

- Reference copying is usually a bad idea (not always but realize what you are doing)
- Deep copying provides maximal protection against aliasing (but takes a lot of time and space if it was not necessary)
- Storage space and time used
  - Reference: least
  - Shallow: middle
  - Deep: most
- If the class is mutable, aliasing is something to be avoided and you must have true copies to prevent privacy leaks and modifications outside.
- If you know the class is immutable, aliasing doesn’t hurt but neither does making true copies (except wasted space and time).
- If storage is an issue, aliasing problems may be worth coping with but must be well documented.