Cloning

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 8

Copying an object

- Normal assignment creates two references to the same object
  - Updates via either name update the object

```
public class Employee {
  private String name;
  private double salary;

  public Employee(String n, double s){
    name = n;
    salary = s;
  }

  public void setname(String n){
    name = n;
  }
}

...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1;
e2.setname("Eknath");  // e1 also updated
```

Copying an object

- Normal assignment creates two references to the same object
  - Updates via either name update the object
- What if we want two separate but identical objects?
  - e2 should be initialized to a disjoint copy of e1

```
public class Employee {
  private String name;
  private double salary;

  public Employee(String n, double s){
    name = n;
    salary = s;
  }

  public void setname(String n){
    name = n;
  }
}

...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1;
e2.setname("Eknath");  // e1 also updated
```

Copying an object

- Normal assignment creates two references to the same object
  - Updates via either name update the object
- What if we want two separate but identical objects?
  - `e2` should be initialized to a disjoint copy of `e1`
- How does one make a faithful copy?

```
public class Employee {
  private String name;
  private double salary;

  public Employee(String n, double s){
    name = n;
    salary = s;
  }

  public void setname(String n){
    name = n;
  }
}

...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1;
e2.setname("Eknath");  // e1 also updated
```

The clone() method

- Object defines a method clone()

```
public class Employee {
  private String name;
  private double salary;

  public Employee(String n, double s){
    name = n;
    salary = s;
  }

  public void setname(String n){
    name = n;
  }
}
```

The clone() method

- Object defines a method clone()
- e1.clone() returns a bitwise copy of e1

```
public class Employee {
  private String name;
  private double salary;

  public Employee(String n, double s){
    name = n;
    salary = s;
  }

  public void setname(String n){
    name = n;
  }
}

...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1.clone();
e2.setname("Eknath");  // e1 not updated
```

The clone() method

- Object defines a method clone()
- e1.clone() returns a bitwise copy of e1
- Why a bitwise copy?
  - Object does not have access to private instance variables
  - Cannot build up a fresh copy of e1 from scratch

```
public class Employee {
  private String name;
  private double salary;

  public Employee(String n, double s){
    name = n;
    salary = s;
  }

  public void setname(String n){
    name = n;
  }
}

...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1.clone();
e2.setname("Eknath");  // e1 not updated
```

The clone() method

- Object defines a method clone()
- e1.clone() returns a bitwise copy of e1
- Why a bitwise copy?
  - Object does not have access to private instance variables
  - Cannot build up a fresh copy of e1 from scratch
- What could go wrong with a bitwise copy?

```
public class Employee {
  private String name;
  private double salary;

  public Employee(String n, double s){
    name = n;
    salary = s;
  }

  public void setname(String n){
    name = n;
  }
}

...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1.clone();
e2.setname("Eknath");  // e1 not updated
```

Shallow copy

- What if we add an instance variable `Date` to `Employee`?
  - Assume `update()` updates the components of a `Date` object

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){
    name = n;
  }

  public void setbday(int dd, int mm, int yy){
    birthday.update(dd,mm,yy);
  }
}
```

Shallow copy

- What if we add an instance variable `Date` to `Employee`?
  - Assume `update()` updates the components of a `Date` object
- Bitwise copy made by `e1.clone()` copies the reference to the embedded `Date`
  - `e2.birthday` and `e1.birthday` refer to the same object
  - `e2.setbday()` affects `e1.birthday`

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){
    name = n;
  }

  public void setbday(int dd, int mm, int yy){
    birthday.update(dd,mm,yy);
  }
}
...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1.clone();
e2.setname("Eknath");  // e1 name not updated
e2.setbday(16,4,1997); // e1 bday updated!
```

Shallow copy

- What if we add an instance variable `Date` to `Employee`?
  - Assume `update()` updates the components of a `Date` object
- Bitwise copy made by `e1.clone()` copies the reference to the embedded `Date`
  - `e2.birthday` and `e1.birthday` refer to the same object
  - `e2.setbday()` affects `e1.birthday`
- Bitwise copy is a shallow copy
  - Nested mutable references are copied verbatim

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){
    name = n;
  }

  public void setbday(int dd, int mm, int yy){
    birthday.update(dd,mm,yy);
  }
}
...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1.clone();
e2.setname("Eknath");  // e1 name not updated
e2.setbday(16,4,1997); // e1 bday updated!
```

Shallow copy

- What if we add an instance variable `Date` to `Employee`?
  - Assume `update()` updates the components of a `Date` object
- Bitwise copy made by `e1.clone()` copies the reference to the embedded `Date`
  - `e2.birthday` and `e1.birthday` refer to the same object
  - `e2.setbday()` affects `e1.birthday`
- Bitwise copy is a shallow copy
  - Nested mutable references are copied verbatim

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){
    name = n;
  }

  public void setbday(int dd, int mm, int yy){
    birthday.update(dd,mm,yy);
  }
}
...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1.clone();
e2.setname("Eknath");  // e1 name not updated
e2.setbday(16,4,1997); // e1 bday updated!
```

Deep copy

- Deep copy recursively clones nested objects

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}
}
```

Deep copy

- Deep copy recursively clones nested objects
- Override the shallow `clone()` from `Object`

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone(){
    Employee newemp =
          (Employee) super.clone()
    Date newbday = birthday.clone();
    newemp.birthday = newbday;
    return newmp;
  }
}
```

Deep copy

- Deep copy recursively clones nested objects
- Override the shallow `clone()` from `Object`
- `Object.clone()` returns an `Object`
  - Cast `super.clone()`

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone(){
    Employee newemp =
          (Employee) super.clone()
    Date newbday = birthday.clone();
    newemp.birthday = newbday;
    return newmp;
  }
}
```

Deep copy

- Deep copy recursively clones nested objects
- Override the shallow `clone()` from `Object`
- `Object.clone()` returns an `Object`
  - Cast `super.clone()`
- `Employee.clone()` returns an `Employee`
  - Allowed to change the return type

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone(){
    Employee newemp =
          (Employee) super.clone()
    Date newbday = birthday.clone();
    newemp.birthday = newbday;
    return newmp;
  }
}
```

Deep copy . . .

- What if `Manager` extends `Employee`?

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone(){...}
}
```

Deep copy . . .

- What if `Manager` extends `Employee`?
- New instance variable `promodate`

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone(){...}
}

public class Manager extends Employee {
  private Date promodate;
  ...
}
```

Deep copy . . .

- What if `Manager` extends `Employee`?
- New instance variable `promodate`
- `Manager` inherits deep copy `clone()` from `Employee`

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone(){...}
}

public class Manager extends Employee {
  private Date promodate;
  ...
}
```

Deep copy . . .

- What if `Manager` extends `Employee`?
- New instance variable `promodate`
- `Manager` inherits deep copy `clone()` from `Employee`
- However `Employee.clone()` does not know that it has to deep copy `promodate`!

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone(){...}
}

public class Manager extends Employee {
  private Date promodate;
  ...
}
```

Deep copy . . .

- What if `Manager` extends `Employee`?
- New instance variable `promodate`
- `Manager` inherits deep copy `clone()` from `Employee`
- However `Employee.clone()` does not know that it has to deep copy `promodate`!
- Cloning is subtle, so Java puts in some restrictions

```
public class Employee {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone(){...}
}

public class Manager extends Employee {
  private Date promodate;
  ...
}
```

Restrictions on `clone()`

- To allow `clone()` to be used, a class has to implement `Cloneable` interface
  - Marker interface

```
public class Employee implements Cloneable {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}
}
...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1.clone();
e2.setname("Eknath");  // e1 not updated
```

Restrictions on `clone()`

- To allow `clone()` to be used, a class has to implement `Cloneable` interface
  - Marker interface
- `clone()` in `Object` is `protected`
  - Only `Employee` objects can `clone()`

```
public class Employee implements Cloneable {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}
}
...
Employee e1 = new Employee("Dhruv", 21500.0);
Employee e2 = e1.clone();
e2.setname("Eknath");  // e1 not updated
```

Restrictions on `clone()`

- To allow `clone()` to be used, a class has to implement `Cloneable` interface
  - Marker interface
- `clone()` in `Object` is `protected`
  - Only `Employee` objects can `clone()`
- Redefine `clone()` as `public` to allow other classes to clone `Employee`
  - Expanding visibility from `protected` to `public` is allowed

```
public class Employee implements Cloneable {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone(){...}
}
```

Restrictions on clone()

- To allow clone() to be used, a class has to implement Cloneable interface
  - Marker interface
- clone() in Object is protected
  - Only Employee objects can clone()
- Redefine clone() as public to allow other classes to clone Employee
  - Expanding visibility from protected to public is allowed
- Object.clone() throws CloneNotSupportedException
  - Catch or report this exception
  - Call clone() in try block

```
public class Employee implements Cloneable {
  private String name;
  private double salary;
  private Date birthday;
  ...
  public void setname(String n){...}

  public void setbday(...){...}

  public Employee clone()
    throws CloneNotSupportedException {...}
}
```

Summary

- Making a faithful copy of an object is a tricky problem
- Java provides a `clone()` function in `Object` that does shallow copy

Summary

- Making a faithful copy of an object is a tricky problem
- Java provides a `clone()` function in `Object` that does shallow copy
- However, shallow copy aliases nested objects

Summary

- Making a faithful copy of an object is a tricky problem
- Java provides a `clone()` function in `Object` that does shallow copy
- However, shallow copy aliases nested objects
- Deep copy solves the problem, but inheritance can create complications

Summary

- Making a faithful copy of an object is a tricky problem
- Java provides a `clone()` function in `Object` that does shallow copy
- However, shallow copy aliases nested objects
- Deep copy solves the problem, but inheritance can create complications
- To force programmers to consciously think about these subtleties, Java puts in some checks to using `clone()`

Summary

- Making a faithful copy of an object is a tricky problem
- Java provides a `clone()` function in `Object` that does shallow copy
- However, shallow copy aliases nested objects
- Deep copy solves the problem, but inheritance can create complications
- To force programmers to consciously think about these subtleties, Java puts in some checks to using `clone()`
- Must implement marker interface `Cloneable` to allow `clone()`
- `clone()` is `protected` by default. override as `public` if needed
- `clone()` in `Object` throws `CloneNotSupportedException`, which must be taken into account when overriding