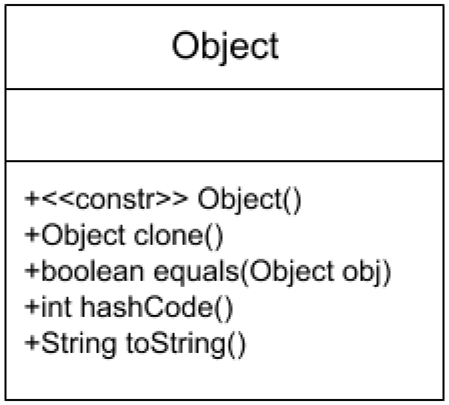
U9L4: `Object` the Ultimate Super Class

the `Object` class

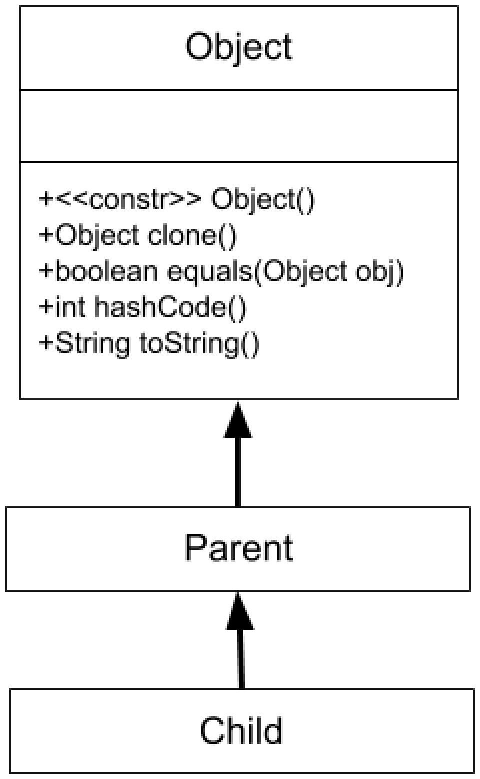
- In Java, every class that doesn't explicitly extend another class, implicitly extends the `Object` class.
- The `Object` class includes several inherited methods that we will look at closer in this lesson.



Does Child have access to Object methods?

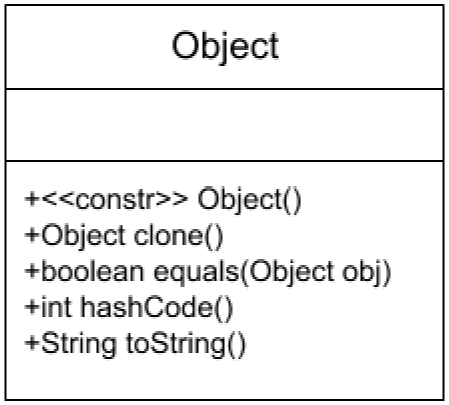
```
public class Parent{
    // implementation not shown
}

public class Child extends Parent{
    // implementation not shown
}
```



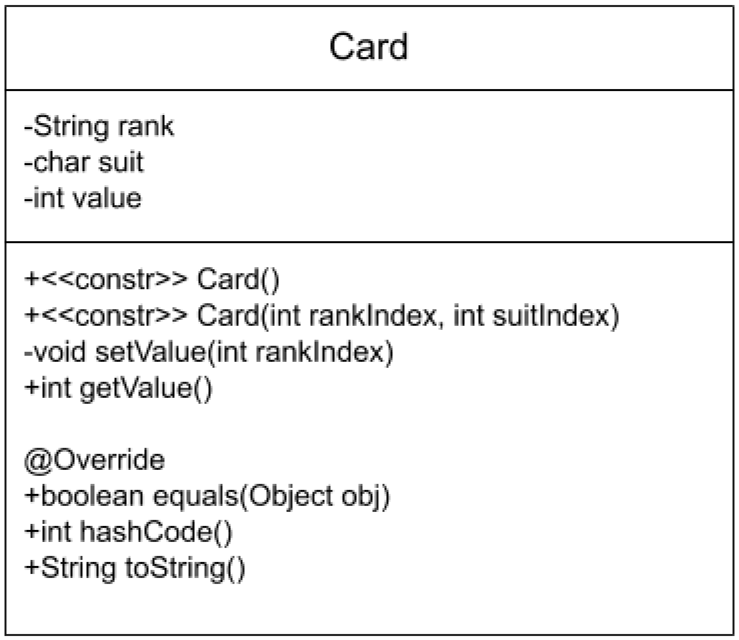
Two Methods to look at in this lesson

- `String toString()`: Returns a **String** representation of an object.
- `boolean equals(Object obj)`: Returns a boolean representing whether two objects are equal or not.



the `Card` class

- `Card.java`: Review the starter code. (Posted to Classroom).
- `CardTest.java`: Create with `main()` method for testing.

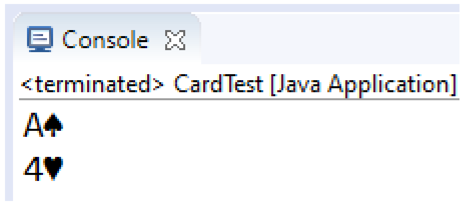


`toString()` method

- Returns a String representation of an object.
- Defaults to printing the reference value.

Discuss benefits of using @Override annotation.

```java
@Override
public String toString(){
    return rank + suit;
}
```

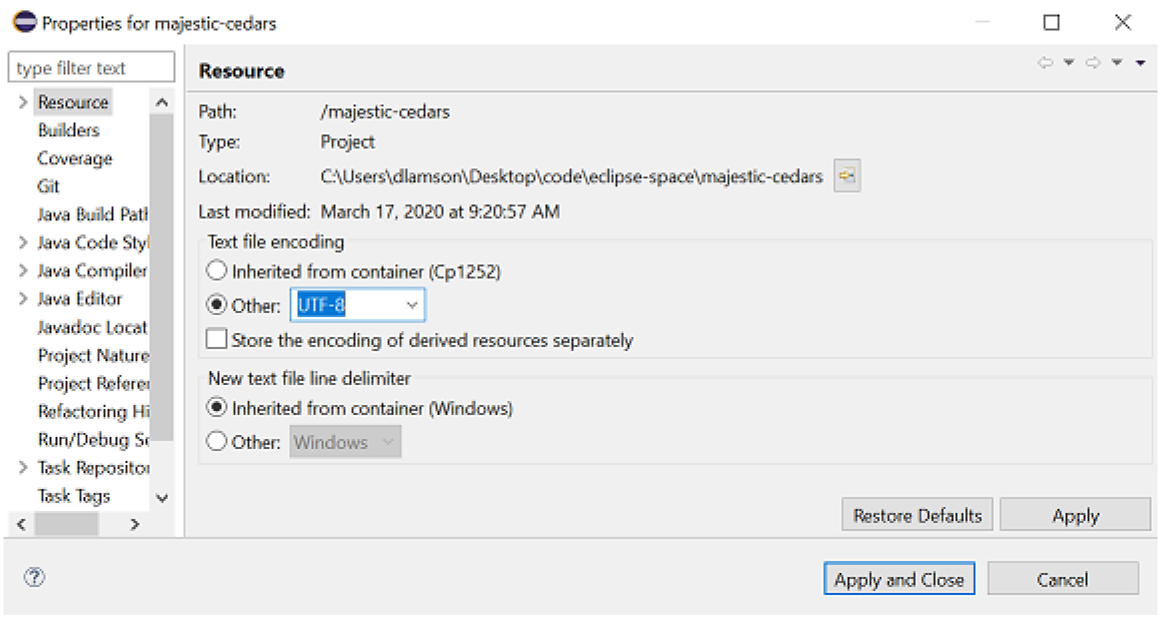


```
<terminated> CardTest [Java Application]
A♠
4♥
```

testing the `toString()`

```java
// creates the Ace of Spades
Card c1 = new Card(12, 0);
// creates a random card
Card c2 = new Card();
System.out.println(c1);
System.out.println(c2);
```

Can't see the card suits?
Go to `Project > Settings` and change encoding to `UTF-8`


Properties for majestic-cedars

Resource

Path: /majestic-cedars
Type: Project
Location: C:\Users\dlamson\Desktop\code\eclipse-space\majestic-cedars
Last modified: March 17, 2020 at 9:20:57 AM

Text file encoding
Inherited from container (Cp1252) [✓ Other: UTF-8]
Store the encoding of derived resources separately

New text file line delimiter
[✓ Inherited from container (Windows)] Other: Windows

`.equals()` method

identity vs. equality

- The intent of an `equals` method is to determine whether two objects are equivalent to each other.
- The intent of the `==` operator is to determine whether two objects share the same identity.
- The default `.equals()` method from the `Object` class only checks for identity. In other words, it is no different than the `==` operator.
- It is up to us as programmers to define what it means for two objects to be *equal* to each other.

`.equals()` method

simplified overload (not a proper override)

- We will define two cards equal if their ranks and suits are the same.
- **NOTE:** This is not a proper override.

Think about this. Why?

```java
public boolean equals(Card other){
    return this.rank.equals(other.rank) && this.suit == other.suit;
}
```

testing the `.equals()` overload

```java
Card c1 = new Card(12, 0);
Card c2 = new Card(12, 0);
// should print false (different references)
System.out.println(c1 == c2);
// should print true (both ace of spades)
System.out.println(c1.equals(c2));
```

the problem with this

- The method in the `Object` class takes an `Object` instance as a parameter.
- See javadoc
- If we use a *polymorphic* reference for one of our playing cards our method won't be called.
- Instead, the default inherited version from the `Object` class will be called.

c2 referenced polymorphically

```
Card c1 = new Card(12, 0);
Object c2 = new Card(12, 0);
System.out.println(c1.equals(c2));
```

- This prints false since c1 and c2 have different references.

properly overriding `.equals()`

We will start by changing the method signature to:

```java
@Override
public boolean equals(Object other)
```

Notice that this will not compile. Java knows that not all `Object` instances are `Card` instances and therefore may not have a *rank* or *suit*. To get around this, we have to **cast** the object as Card.

```java
@Override
public boolean equals(Object other){
    Card otherCard = (Card) other;
```

NOTE: Casting doesn't *do* anything. It just is our way of telling the Java compiler that we know what we are doing, and we will ensure that `other` is a valid `Card` object.

ClassCastExceptions

We have now opened ourselves up to a potential issue by doing this. Since all references are related to the `Object` class, it is now possible to pass practically anything we want to this method. If `other` isn't actually a `Card` instance a `ClassCastException` will be generated at runtime.

```
Card c1 = new Card();
String c2 = "Ace of Spades";

// Will crash at runtime
System.out.println(c1.equals(c2));
```

Why doesn't the compiler catch this?

The `instanceof` operator

- The `instanceof` operator can be used to check for an **is-a** relationship.
- It will return `true` as long as the type is above it in the class hierarchy.

demo instanceof

```
Card c1 = new Card();
String c2 = "Ace of Spades";
System.out.println(c1 instanceOf Card) // true c1 is a Card
System.out.println(c1 instanceOf String) // false c1 is NOT a String
System.out.println(c1 instanceOf Object) // true c1 is an Object
System.out.println(c2 instanceOf Card) // false c2 is NOT a Card
System.out.println(c2 instanceOf String) // true c2 is a String
System.out.println(c2 instanceOf Object) // true c2 is an Object
```

`.equals()` method complete

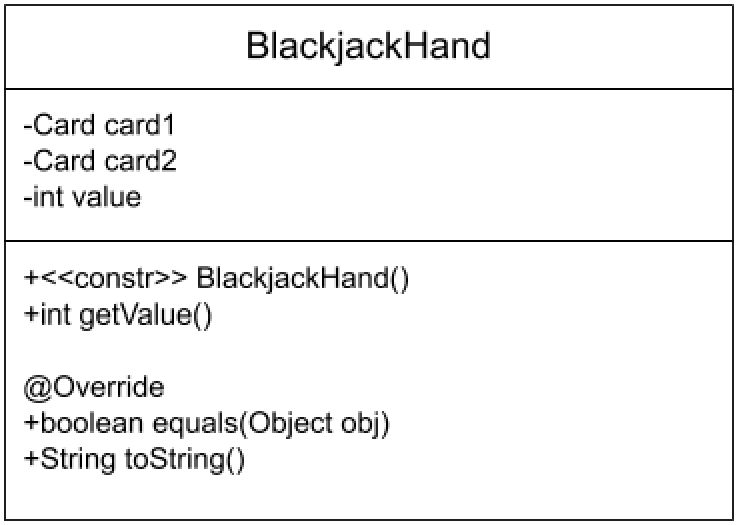
To prevent this we can use the `instanceOf` operator to check if the object actually is a `Card` before trying to cast it as one. If it's not a Card, we will simply return `false` and avoid the runtime error.

```
public boolean equals(Object other){
    if(!(other instanceOf Card)){
        return false;
    }

    Card otherCard = (Card) other;
    return this.rank.equals(otherCard.rank) && this.suit == otherCard.suit;
}
```

LAB-032: Part A - BlackjackHand Class

Write a class `BlackjackHand` class with the following design.



LAB-032: Part A - Continued

- `BlackjackHand()`: The constructor must initialize `card1` and `card2` using the random `Card()` constructor. The value of the hand will be set by adding the values of the cards except when the sum is `22` (both cards are Aces). In this case the value of the hand must be set to `12`.

- `toString()`: A string representation of the hand must show both cards and the value of the hand in an arrangement of your choice.

- `equals()`: Must return `true` if and only if the value of the hands are equal. This must be a proper override similar to the `Card` class.

- Submit this file to Classroom.

LAB-032: Part B - Simulation

Use this BlackjackHand Class to run the following simulations. Simulation results will be entered into a Google Form. Do not submit your simulation code.

1. Generate a player and dealer hand 1 million times. Use your `.equals()` method to determine how many times the dealer and player were dealt hands with equal values. (Enter in Google Form)

2. The possible values of starting hands ranges from 4 to 21. Generate 1000000 hands and track how many times each of these values occurred in the simulation. (Enter in Google Form)