Introduction to Enumerations

CMSC 202

Enumerated Values

- Enumerated values are used to represent a set of named values.
- Historically in Java (and other languages), these were often stored as constants.
- For example, in Java . . .

```
public static final int SUIT_CLUBS = 0;
public static final int SUIT_DIAMONDS = 1;
public static final int SUIT_HEARTS = 2;
public static final int SUIT_SPADES = 3;
```

Issues with this Approach

- There are, however, a number of issues with this approach.
  - Acceptable values are not obvious
  - No type safety
  - No name-spacing
  - Not printable

Acceptable Values Not Obvious

- Since the values are just integers, it's hard at a glance to tell what the possible values are.
- Take this method from swing's JLabel class.

```
public void setHorizontalAlignment(int alignment) {
    /* ... */
}
```

- Any clue as to what the valid values are for the alignment parameter?
  - Have to resort to reading the documentation

No Type Safety

- Since the values are just integers, the compiler will let you substitute any valid integer.
- For example, there's nothing stopping one from passing in 1, -3, or 438523423 into the following method.

```
public void drawSuitOnCard(int suit) {
    /* ... */
}
```

- There's no way to constrain to only "suit" ints.

No Name-Spacing

- With our card example, we prefixed each of the suits with “SUIT_” .
- We chose to prefix all of those constants with this prefix to potentially disambiguate from other enumerated values of the same class.
- For example, had we chosen to also enumerate the card faces (e.g. Jack, Queen, …) we would want to make it clear that they were representing the card faces.
  - For example, we might have “FACE_ACE”.

Not Printable

- Since they are just integers, if we were to print out the values, they'd simply display their numerical value.
- Similar problem as when reading the method parameters
  - Need to consult the documents to decipher values

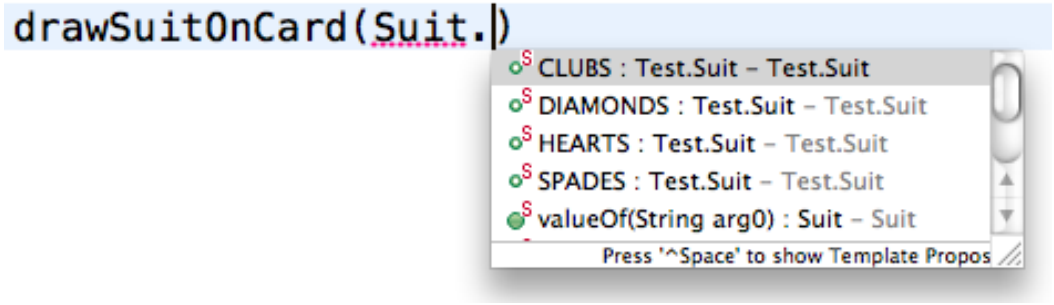
Enums to the Rescue

- Java 5 added an ***enum type*** to the language.
- Declared using the **enum** keyword instead of class
- In its simplest form, it contains a comma-separated list of names representing each of the possible options.

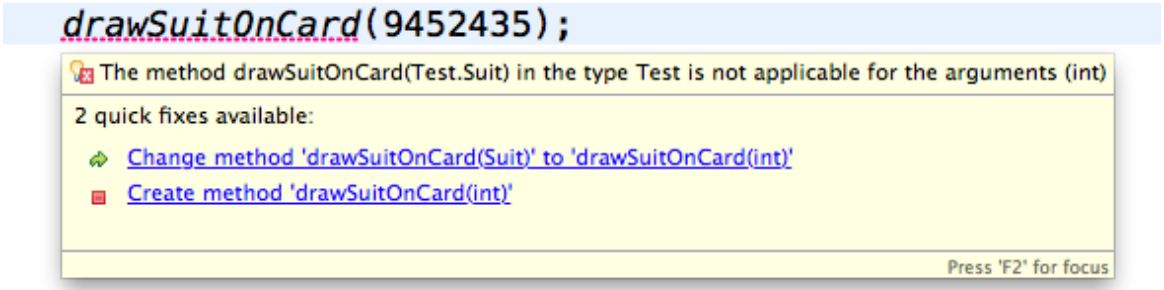
```
public enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }
```

Enums Address These Issues

- Acceptable values are now obvious — must choose one of the Suit enumerated values…

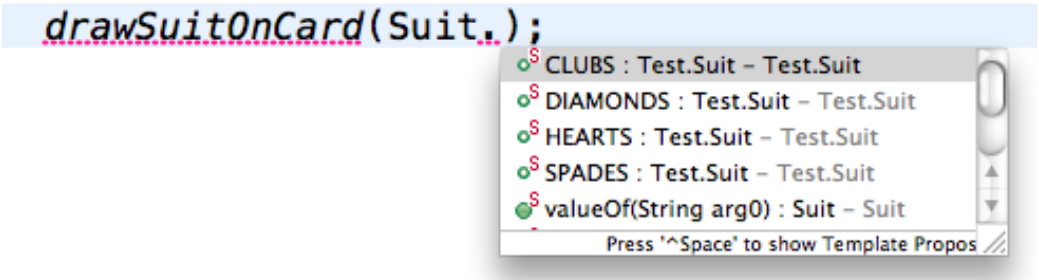


- Type safety — possible values are enforced by the compiler



Enums Address These Issues

- Every value is name-spaced off of the enum type itself.



- Printing the enum value is actually readable.

```
System.out.print("Card is a Queen of " + Suit.HEARTS);
// Prints "Card is a Queen of HEARTS"
```

Additional Benefits

- Storage of additional information
- Retrieval of all enumerated values of a type
- Comparison of enumerated values

Storage of Additional Information

- Enums are objects
- So they can have…
  - Member variables
  - Methods
- For example…
  - We could embed the color of the suit within the Suit.
  - We can then read the value using a getter, etc.

```
public enum Suit {

  CLUBS(Color.BLACK),
  DIAMONDS(Color.RED),
  HEARTS(Color.RED),
  SPADES(Color.BLACK);

  private Color color;

  // Java will prevent construction
  // outside of enum declaration
  Suit(Color c) {
    this.color = c;
  }

  public Color getColor() {
    return this.color;
  }
}
```

Retrieval of All Enumerated Values

- All enum types will automatically have a *values()* method that returns an array of all enumerated values for that type.

```
Suit[ ] suits = Suit.values();
for(Suit s : suits) {
        System.out.println(s);
}
```

Comparison of Enumerated Values

- Since users cannot construct enum instances, there can only be one instance of each value.
- As such, we can actually compare enums using the **==** operator.

```
if(suit == Suit.CLUBS) {
        // do something
}
```

Comparison of Enumerated Values

- Enums can also be used with the **switch** control structure.

```
Suit suit = /* ... */;

switch (suit) {
        case CLUBS:
        case SPADES:
                // do something
                break;
        case HEARTS:
        case DIAMONDS:
                // do something else
                break;
        default:
                // yet another thing
                break;
}
```

One Gotcha

- If you have a reference to an enum instance, then you're assured to have a valid value.
- The key word being "if" — you'll likely still need to check that the reference is set.
  - In other words, you may need to check that the reference does/doesn't refer to null.