

Roulette: Inheritance Case Study

- **Roulette involves a player, a wheel, and bets**
 - Real game has several players, we'll use one
 - Real game has lots of kinds of bets, we'll use three but make it simple to add more
- **Instead of brainstorming classes, we'll take as given:**
 - **Wheel**
 - **Bet**
 - **Bankroll**
 - **Roulette (game)**
 - **What's missing?**
 - **What are responsibilities, collaborations?**

What are scenarios?

- **User/player given choice of bet**
 - **Bet choice made**
 - **Wager made**
 - **Wheel spins**
 - **Payoff given**
 - **Play again?**

- **What happens when player bets?**
 - **What's recorded?**
 - **How is winning determined?**
 - **What about multiple players, multiple bets?**

What is a bet?

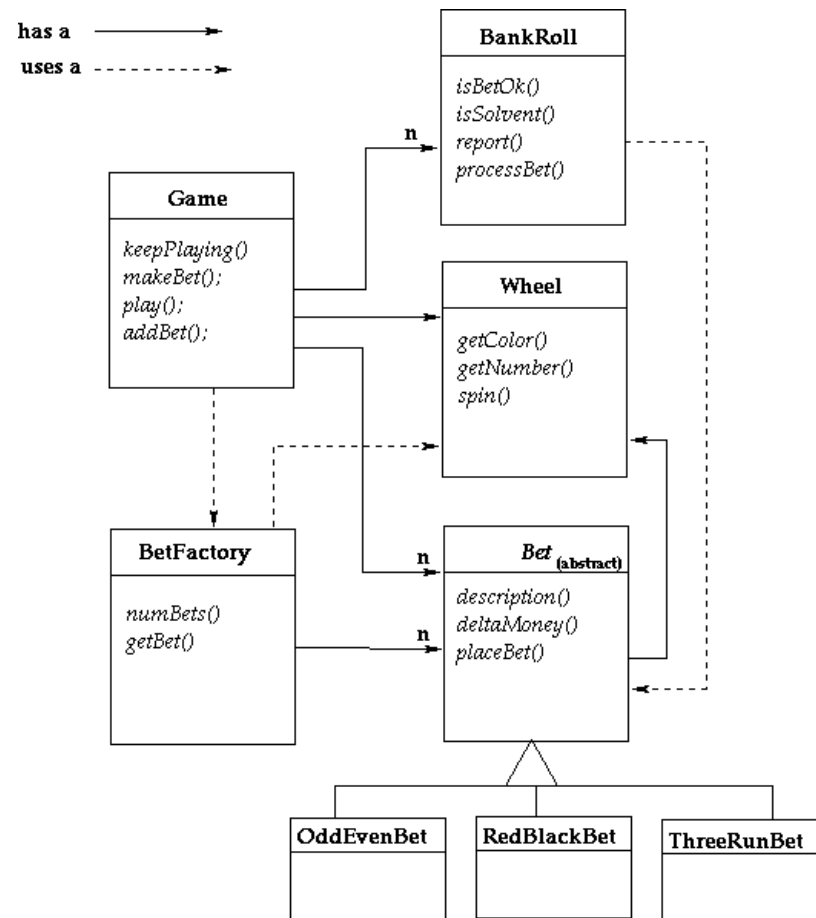
- **Difference between wager and bet**
 - **Bet contains wager amount**
 - **Different bets have different payoffs**
- **What happens after the wheel rolls and payoff occurs?**

```
if (myBet == redblack) ...  
else if (myBet == oddeven) ...
```

- **Problems with this?**
- **Open closed principle?**
- **Canonical OO tenet: avoid chains of if/else (when you can)**

Roulette Class Diagram

- **Has-a relationship**
 - **Bet has a wheel, how/why?**
 - **BetFactory has bets**
 - What bets?
 - **Game has a bankroll**
 - Does this make sense?
- **Uses-a relationship**
 - **Parameters**
 - **Return values**
 - **Local variables**
- **Has-a relationship**
 - **Inheritance, use-as-a**
 - **Substitutable for-a**



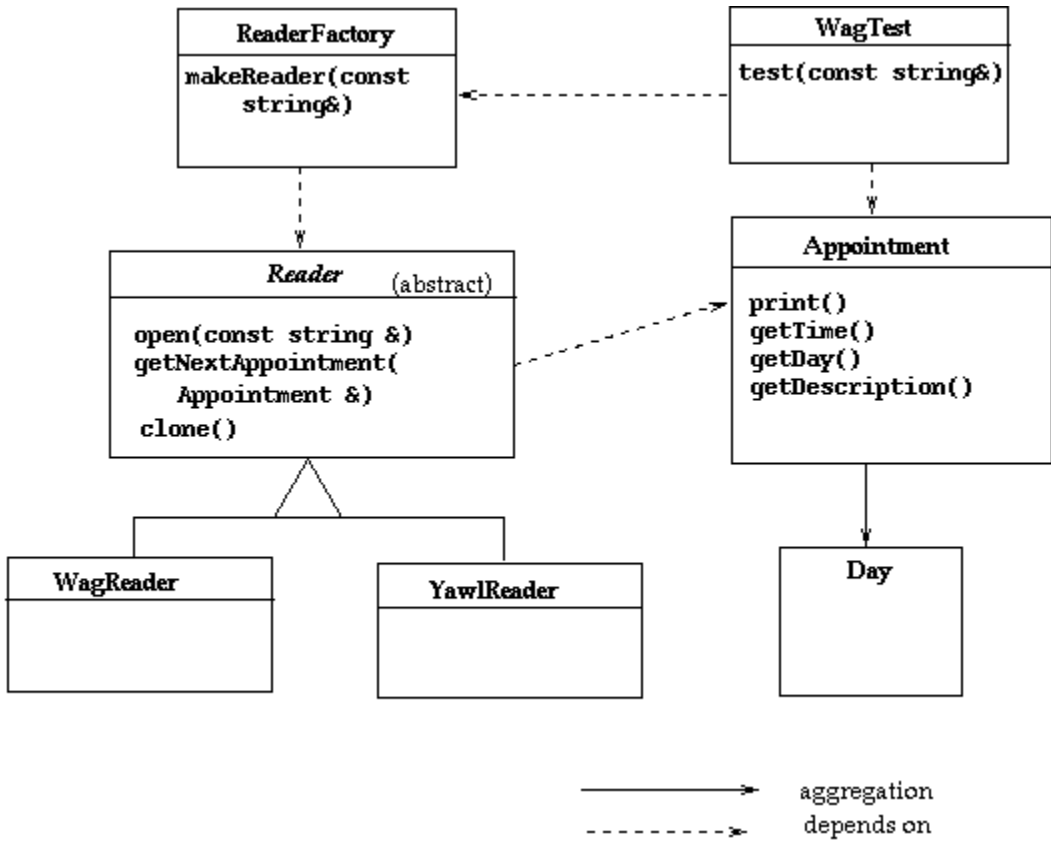
Inheritance guidelines in C++

- **Inherit from Abstract Base Classes (ABC)**
 - one pure virtual function needed (=0)
 - must have virtual destructor implemented
 - can have pure virtual destructor implemented, but not normally needed
- **Avoid protected data, but sometimes this isn't possible**
 - data is private, subclasses have it, can't access it
 - keep protected data to a minimum
- **Single inheritance, assume most functions are virtual**
 - multiple inheritance ok when using ABC, problem with data in super classes
 - virtual: some overhead, but open/closed principle intact

Designing hyperwag

- **Keep classes small and cohesive**
 - as simple as possible, but no simpler
 - member functions should also be small
- **Design for change**
 - specifications, requirements, design
 - example: other formats for table design?
- **Design first, code second, but revisit design**
- **Know the language, but don't let the language rule the design**
- **Get the classes right, concentrate on what *not* how**

One view of hyperwag



Patterns: Abstract Factory

- **Abstract Factory/Factory aka “kit”**
 - system should be independent of how products created
 - system should be configured with one of multiple products (or families of products, e.g., Win95, Motif)
 - you want to provide a class library of products and reveal interfaces but not implementations
- **Consequences**
 - factory encapsulates responsibility and process of creating objects, clients only see abstract interface. “Real names” hidden in factory
 - supporting new products can be difficult depending on the situation (but see Prototype pattern)
- **Often want only one factory accessible in a program**

Patterns: Singleton

- **Singleton**
 - enforce exactly one instance of a class, accessible in a well-defined manner
 - possible to extend via inheritance
- **Consequences**
 - controlled access to single instance, e.g.,

```
Foo * foo = Foo::getInstance();
```
 - no global variables
 - no need to rely solely on class/static functions
- **Implementation**
 - private constructor
 - getInstance (or other class/static functions, see hyperwag)

Patterns: Prototype

- **Use a prototypical instance to clone new objects**
 - classes used in a program can be specified/loaded at runtime
 - avoid hierarchy of factories that parallels hierarchy of classes
- **Abstract Prototype class implements `clone()`**
 - how to copy? deep vs. shallow
 - how to initialize clones in subclasses
- **Managing Prototypes**
 - use a factory or a prototype manager with registered prototypes
 - cloning can be tough (e.g., circular references)