**1. Во что компилируются анонимные внутренние классы?**  
Анонимные внутренние классы компилируются в файлы *внешнийКласс$n.class*. На месте внешнего класса, соответственно, название обрамляющего класса, внутри которого описывается анонимный внутренний класс. На месте n число от 1 до количества анонимных классов.   
  
**2. Можно ли наследовать внутренние классы?**  
Наследовать внутренние классы от других — можно.   
Наследование от внутреннего класса получается чуть сложнее, чем обычное, так как конструктор внутреннего класса связывается со ссылкой на окружающий внешний объект. Проблема состоит в том, что «скрытая» ссылка на объект объемлющего внешнего класса должна быть инициализирована, а в производном классе больше не существует объемлющего объекта по умолчанию. Для явного указания объемлющего внешнего объекта применяется специальный синтаксис:

//: innerclasses/InheritInner.java  
// Наследование от внутреннего класса.  
   
class WithInner {  
  class Inner {}  
}  
   
public class InheritInner extends WithInner.Inner {  
  //! InheritInner() {} // He компилируется   
  InheritInner(WithInner wi) {  
    wi.super();  
  }  
  public static void main(String[] args) {  
    WithInner wi = new WithInner();  
    InheritInner ii = new InheritInner(wi);  
  }  
}

Здесь класс *InheritInner* расширяет только внутренний класс, а не внешний. Но когда дело доходит до создания конструктора, предлагаемый по умолчанию конструктор не подходит, и вы не можете просто передать ссылку на внешний объект. Необходимо включить в тело конструктора выражение

ссылкаНаОбъемлющийКласс.super();

в теле конструктора. Оно обеспечит недостающую ссылку, и программа откомпилируется.  
  
**3. Можно ли наследовать анонимные внутренние классы?**  
Описывая анонимный класс мы уже наследуемся от какого-то класса или реализуем какой-либо интерфейс. К анонимным классам напрямую нельзя применить слова extends или implements, но ведь никто не мешает заранее подготовиться и расширить нужный интерфейс, который будем реализовывать с помощью анонимного класса. Пример в коде ниже.

import java.awt.event.WindowListener;  
  
public class TestExtendAnonym {  
    private interface MyInterface extends Runnable, WindowListener {  
    }  
  
    Runnable r = new MyInterface() {  
    ...  
    //Пример того как реализовать 2 и более интерфейса в анонимном классе  
    };  
}

Наследоваться от анонимного класса нельзя.   
  
**4. Можно ли переопределять внутренние классы?**  
Переопределение внутреннего класса, как если бы он был еще одним методом внешнего класса, фактически не имеет никакого эффекта:

//: innerclasses/BigEgg.java  
// Внутренний класс нельзя переопределить   
// подобно обычному методу,  
import static net.mindview.util.Print.\*;  
   
class Egg {  
  private Yolk y;  
  protected class Yolk {  
    public Yolk() { print("Egg.Yolk()"); }  
  }  
  public Egg() {  
    print("New Egg()");  
    y = new Yolk();  
  }  
}         
   
public class BigEgg extends Egg {  
  public class Yolk {  
    public Yolk() { print("BigEgg.Yolk()"); }  
  }  
  public static void main(String[] args) {  
    new BigEgg();  
  }  
}

Вывод:

New Egg()  
Egg.Yolk()

Конструктор по умолчанию автоматически синтезируется компилятором, а в нем вызывается конструктор по умолчанию из базового класса. Можно подумать, что при создании объекта *BigEgg* должен использоваться «переопределенный» класс *Yolk*, но это отнюдь не так, как видно из результата работы программы.  
  
Этот пример просто показывает, что при наследовании от внешнего класса ничего особенного с внутренними классами не происходит. Два внутренних класса — совершенно отдельные составляющие, с независимыми пространствами имен.   
Иными словами нельзя.   
  
**5. Какие ограничения есть у локальных классов?**  
Вначале вспомним что такое локальный класс. Это класс, описанный в блоке кода, то есть, по-простому — между кавычек {}. Наиболее часто эти кавычки являются телом метода. Но могут они быть и просто блоком, статическим блоком, телом if-ов, циклов и т.д.  
Локальный класс наделён особенностями внутренних классов, но имеет отличительные черты, а именно:

1. он имеет доступ только к финальным полям и аргументам обрамляющего метода, а также ко всем полям обрамляющего класса, в том числе приватным и статическим;
2. локальный класс виден и может создаваться только в блоке, в котором описан;
3. у локального класса не ставиться модификатор доступа;
4. не может иметь статических полей, методов, классов (за исключением финальных);
5. локальный класс, объявленный в статическом блоке может обращаться только к статическим полям внешнего класса.

**Но!** Начиная с Java8 мы можем обращаться в локальных классах к не финальным локальным переменным, если они не были изменены до момента инициализации класса. Также теперь стало возможным обращение к не финальным параметрам метода.  
  
**6. Может ли анонимный внутренний класс содержать статические методы?**  
Нет. У Анонимных внутренних классов, как и у внутренних классов не может быть статических полей, методов. (вспомним, что анонимные классы компилируются в обычные внутренние, а те, в свою очередь, связаны с объектом обрамляющего класса)  
  
**7. Можно ли создать объект внутреннего класса, если у внешнего класса только private конструктор?**  
Имея подобный код:

public class PrivateConst {private PrivateConst() {}

public class InnerClass{public void f(){System.out.println("hello");}}}

Напрямую, в другом классе (вне обрамляющего), конечно, создать объект *InnerClass* следующим способом не получится:

PrivateConst.InnerClass priv = new PrivateConst().new InnerClass();

Но! Что если у нас есть метод, возвращающий экземпляр PrivateConst:

public class PrivateConst {

private static PrivateConst instance;

private PrivateConst() {}

public static PrivateConst getInstance() {

instance = new PrivateConst();

return instance;

}

public class InnerClass{}

}

В этом случае приватный конструктор нам не помеха для создания объекта *InnerClass*. Так же мы без проблем сможем создавать его в методах и в других внутренних классах, принадлежащих *PrivateConst*. Ответ — можно, если каким-либо способом нам удастся получить объект обрамляющего класса.  
  
**8. Можно ли объявлять внутренние классы private?**  
Да, можно.  
  
PS Обоснования так и не нашел, но на философии java встречались подобные примеры.   
Плюс IDE не ругается.   
Буду признателен за обоснование, но предположу, что в этом плане внутренний класс ничем не отличается от обычного класса.   
  
**9. Можно ли объявлять анонимные внутренние классы private?**  
Аналогично (в плане не нашел обоснования). Можно объявить private переменную от типа которой наследуется наш анонимный класс.   
  
**10. Сколько у класса максимально может быть внутренних классов?**  
Сколь угодно много. Ограничение особенности ОС и длинны имени файлов.