```
# Revision:

def save(self, *args, **kwargs):
    if self.price < 5:
        self.is_discounted = True
    else:
        self.is_discounted = False

    super.save(*args, **kwargs)


#1.
class BaseCharacter(models.Model):
    name = models.CharField(
        max_length=100,
    )
    description = models.TextField()

    class Meta:
        abstract = True


class Mage(BaseCharacter):
    elemental_power = models.CharField(
        max_length=100,
    )
    spellbook_type = models.CharField(
        max_length=100,
    )


class Assassin(BaseCharacter):
    weapon_type = models.CharField(
        max_length=100,
    )
    assassination_technique = models.CharField(
        max_length=100,
    )


class DemonHunter(BaseCharacter):
    weapon_type = models.CharField(
        max_length=100,
    )
    demon_slaying_ability = models.CharField(
        max_length=100,
    )


class TimeMage(Mage):
    time_mage_mastery = models.CharField(
        max_length=100,
    )
    temporal_shift_ability = models.CharField(
        max_length=100,
    )


class Necromancer(Mage):
```

```
    raise_dead_ability = models.CharField(
        max_length=100,
    )


class ViperAssassin(Assassin):
    venomous_strikes_mastery = models.CharField(
        max_length=100,
    )
    venomous_bite_ability = models.CharField(
        max_length=100,
    )


class ShadowbladeAssassin(Assassin):
    shadowstep_ability = models.CharField(
        max_length=100,
    )


class VengeanceDemonHunter(DemonHunter):
    vengeance_mastery = models.CharField(
        max_length=100,
    )
    retribution_ability = models.CharField(
        max_length=100,
    )


class FeldbladeDemonHunter(DemonHunter):
    feldblade_ability = models.CharField(
        max_length=100,
    )


#2.
class UserProfile(models.Model):
    username = models.CharField(
        max_length=70,
        unique=True,
    )
    email = models.EmailField(
        unique=True,
    )
    bio = models.TextField(
        null=True,
        blank=True,
    )


class Message(models.Model):
    sender = models.ForeignKey(
        to=UserProfile,
        related_name='sent_messages',
        on_delete=models.CASCADE,
    )
    receiver = models.ForeignKey(
        to=UserProfile,
        related_name='received_messages',
```

```
        on_delete=models.CASCADE,
    )
    content = models.TextField()
    timestamp = models.DateTimeField(
        auto_now_add=True,
    )
    is_read = models.BooleanField(
        default=False,
    )

    def mark_as_read(self):
        self.is_read = True

    def mark_as_unread(self):
        self.is_read = False

    def reply_to_message(self, reply_content, receiver):
        return Message(
            sender=self.receiver,
            receiver=receiver,
            content=reply_content,
        )

    def forward_message(self, sender, receiver):
        return Message(
            content=self.content,
            sender=sender,
            receiver=receiver,
        )

#3.
class StudentIDField(models.PositiveIntegerField):
    def to_python(self, value):
        try:
            return int(value)
        except ValueError:
            pass


class Student(models.Model):
    name = models.CharField(
        max_length=100,
    )
    student_id = StudentIDField()

#4.
class MaskedCreditCardField(models.CharField):
    def __init__(self, *args, **kwargs):
        kwargs['max_length'] = 20
        super().__init__(*args, **kwargs)

    def to_python(self, value):
        if not isinstance(value, str):
            raise ValidationError("The card number must be a string")

        if not all(d.isdigit() for d in value):
            raise ValidationError("The card number must contain only digits")

        if len(value) != 16:
```

```
            raise ValidationError("The card number must be exactly 16 characters
long")

        return f"****-****-****-{value[-4:]}"


class CreditCard(models.Model):
    card_owner = models.CharField(
        max_length=100,
    )
    card_number = models.MaskedCreditCcardFiled()


#5.

class Hotel(models.Model):
    name = models.CharField(
        max_length=100,
    )
    address = models.CharField(
        max_length=200,
    )


class Room(models.Model):
    hotel = models.ForeignKey(
        to='Hotel',
        on_delete=models.CASCADE,
    )
    number = models.CharField(
        unique=True,
        max_length=100,
    )
    capacity = models.PositiveIntegerField()
    total_guests = models.PositiveIntegerField()
    price_per_night = models.DecimalField(
        max_digits=10,
        decimal_places=2,
    )

    def clean(self):
        if self.total_guests > self.capacity:
            raise ValidationError("Total guests are more than the capacity of the
room")

    def save(self, *args, **kwargs):
        self.clean()
        super.save(*args, **kwargs)

        return f'Room {self.number} crested successfully'


class BaseReservation(models.Model):
    class Meta:
        abstract=True

    room = models.ForeignKey(
        to=Room,
        on_delete=models.CASCADE,
```

```
    )
    start_date = models.DateField()
    end_date = models.DateField()

    def reservation_period(self):
        return (self.end_date - self.start_date).days

    def calculate_total_cost(self):
        total_cost = self.reservation_period()*self.room.price_per_night

        return round(total_cost, 1)

    @property
    def is_available(self):
        reservations = self.__class__.objects.filter(
            room=self.room,
            start_date__lte=self.end_date,
            end_date__gte=self.start_date,
        )
        return not reservations.exists()

    def clean(self):
        if self.start_date >= self.end_date:
            raise ValidationError('Start date cannot be after or in the same end
date')
        if not self.is_available:
            raise ValidationError(f'Room {self.room.number} cannot be reserved')


class RegularReservation(BaseReservation):
    def save(self, *args, **kwargs):
        super().clean()
        super().save(*args, **kwargs)

        return f'Regular reservation for room {self.room.number}'


class SpecialReservation(BaseReservation):
    def save(self, *args, **kwargs):
        super().clean()
        super().save(*args, **kwargs)

        return f'Special reservation for room {self.room.number}'

    def extend_reservation(self, days):
        reservations = SpecialReservation.objects.filter(
            room=self.room,
            start_date__lte=self.end_date + timedelta(days=days),
            end_date__gte=self.start_date,
        )

        if reservations:
            raise ValidationError('Error during extending reservation')
        self.end_date += timedelta(days=days)
        self.save()
        return f'Extend reservation for room {self.room.number} with {days} days'
```